Estimating User Interaction Probability for Non-Guaranteed Display Advertising

A thesis submitted in partial fulfilment of the requirements for the Degree of
Master of Science in Computational and Applied Mathematics

by Alan Williams

Department of Mathematics and Statistics
University of Canterbury

2014

[This page intentionally left blank]

Abstract

Billions of advertisements are displayed to internet users every hour, a market worth approximately $110 billion in 2013. The process of displaying advertisements to internet users is managed by advertising exchanges, automated systems which match advertisements to users while balancing conflicting advertiser, publisher, and user objectives. Real-time bidding is a recent development in the online advertising industry that allows more than one exchange (or demand-side platform) to bid for the right to deliver an ad to a specific user while that user is loading a webpage, creating a liquid market for ad impressions. Real-time bidding accounted for around 10% of the German online advertising market in late 2013, a figure which is growing at an annual rate of around 40%. In this competitive market, accurately calculating the expected value of displaying an ad to a user is essential for profitability.

In this thesis, we develop a system that significantly improves the existing method for estimating the value of displaying an ad to a user in a German advertising exchange and demand-side platform. The most significant calculation in this system is estimating the probability of a user interacting with an ad in a given context. We first implement a hierarchical main-effects and latent factor model which is similar enough to the existing exchange system to allow a simple and robust upgrade path, while improving performance substantially. We then use regularized generalized linear models to estimate the probability of an ad interaction occurring following an individual user impression event. We build a system capable of training thousands of campaign models daily, handling over 300 million events per day, 18 million recurrent users, and thousands of model dimensions. Together, these systems improve on the log-likelihood of the existing method by over 10%.

We also provide an overview of the real-time bidding market microstructure in the German real-time bidding market in September and November 2013, and indicate potential areas for exploiting competitors' behaviour, including building user features from real-time bid responses. Finally, for personal interest, we experiment with scalable k-nearest neighbour search algorithms, nonlinear dimension reduction, manifold regularization, graph clustering, and stochastic block model inference using the large datasets from the linear model .

[This page intentionally left blank]

Acknowledgements

This project was undertaken in association with the Christchurch-based development arm of a European online advertising exchange. The company involved welcomed me into their office and treated me as one of their employees for the duration of the project, going far beyond any obligations in giving me access to their expertise and resources. I am very grateful for this.

During the course of this project I have learnt a great deal about both mathematics and the tools and processes used for commercial software development. My knowledge of distributed computing systems, databases, and high performance computing is due to the input of many people. Special thanks are due to:

- Scott Noakes, who took me on and had thought I could do something useful from the start.
- Ben Baldwin, for being cheerful, generous with his time and resources, and helpful even when under pressure.
- Thorsten Castor, for his professionalism, willingness to assist, and technical ability.
- The company as a whole, for generously continuing to provide financial backing beyond the point to which this project directly benefited the firm.
- Joerg Riechardt, for being willing to spend time in helpful and encouraging discussions with an intern.
- My family and friends for making life fun and providing many opportunities to leave the computer.

[This page intentionally left blank]

Contents

Chapter 1

Online advertising and real-time bidding

This chapter provides necessary background information on the current online advertising industry, advertising exchanges, and demand-side platforms. We then introduce the industrial problem motivating this thesis, and briefly outline the chapters of this thesis.

1.1 A brief history

John Wanamaker, a US department store merchant, stated around 1900 that *half the money I spend on advertising is wasted; the trouble is, I don't know which half.* A little over a century since this statement, billions of dollars of products and services are now purchased annually over the internet, and most people in developed economies use the internet daily. The size of this market, the breadth of the audience, and the ability to record information present advertising opportunities that John Wanamaker could only dream about.

Online display advertising is the process of displaying advertisements to internet users as they interact with website content, use online applications, or consume media content on a PC or mobile device. This medium allows marketers to actively target likely customers and measure return on investment, as well as raising general brand awareness translating to in-store activity [28]. An opportunity to present an advertisement to a user while they are interacting with online content is called an ad impression.

Internet display advertising evolved from newspaper and magazine advertising in the early 1990s, as early website publishers sought to earn revenue from the new medium. Publishers negotiated contracts directly with advertisers to place advertisements alongside their content on their web pages, and advertisers paid the publisher a set fee for each time the page was loaded by a user (called an impression).

Between the early and mid 1990s, the rapid increase in the number of internet users, websites, and companies desiring to advertise online led to rise of firms providing online advertising services. One of the first such firms was DoubleClick, founded in 1996. DoubleClick negotiated banner ad placement with publishers on behalf of the advertisers, measured ad performance, and performed optimization of ad performance and publisher revenue by manually moving ads from low-performing websites to higher-performing websites. Coming into 2014, the quantity of display ads on the internet is orders of magnitude larger than in the 1990s and continues to increase dramatically. The process of delivering ads and optimizing performance and revenue has become highly automated through 'platforms', or technical service providers in the online advertising industry. Online advertising exchanges are a significant type of platform. Exchanges programmatically match available user impressions to available advertisements.

Search advertising (such as advertisements shown among the results returned from search engines) and social media advertising (such as content shared on Facebook and twitter) also forms a significant fraction of the online advertising market today. However, while concepts in this thesis are applicable to these areas, only display advertising is directly addressed.

1.2 Advertising exchanges

Large publishers sell a guaranteed number of user impressions, called premium inventory, directly to high-paying advertisers. The remainder, and the long tail of non-premium inventory from smaller publishers is sold in automated spot markets through brokerages called exchanges. This inventory is often referred to as non-guaranteed inventory. In order to buy inventory on these spot markets, advertisers register ad campaigns with exchanges, along with details such as the desired publishers (if any) and the value they are willing to pay for a user impression. The exchange then matches the supply of user impressions from the publishers with demand from advertisers and organises payment, taking a percentage for itself. A medium-sized exchange can manage the placement of more than tens of thousands of advertisements per second.

1.2.1 Pricing models

Advertisers pay for user impressions using one of two general pricing models. In the first, advertisers pay a low fee per thousand impressions, called cost per mille (CPM). This pricing model is becoming less common, as the price of CPM advertising has historically been artificially high due to unjustified expectations of online advertising's efficacy, and user engagement with non-guaranteed online advertising is seen to be decreasing as the market becomes more saturated and user traffic centres around large platforms such as Facebook. The market is therefore moving to performance-based pricing models.

In performance-based pricing models, advertisers only pay for an ad impression if the user takes a subsequent action providing revenue to the advertiser (e.g. purchasing products and services) that is fully or partially attributable to the ad view - the primary desired outcome of advertising. This pricing model is termed cost-per-action or CPA. However, it is difficult to perform automated attribution of both offline and online purchases to ads, due to the problems inherent in continuously tracking user behaviour in the hours and days following an ad view, and the extremely low rates of occurrences of these actions (e.g. 10^{-6}). Therefore, clicks on ads or engagement with the advertiser's own website are often used as a proxy for user purchases, as they are simple to measure and have higher occurrence rates (e.g. 10^{-4} to 10^{-3}). This is termed cost-per-click, or CPC pricing. It is worth noting that clicks on ads in particular are a spectacularly bad proxy for user purchase intent [42], are easy to game by botnets, and are only used due to their ease of use compared to alternatives. In this thesis, we use the term interaction for generality while noting that clicks are the specific interactions considered.

1.2.2 The exchange auction process

When a user loads a publisher's web page, the publisher requests an advertisement from the exchange it is integrated with. The exchange selects the 'best' advertisement for the user and context from the campaigns registered by advertisers and serves this ad to the user. As this process occurs while the page is loading, the user typically does not even realise it has happened. The selection of the 'best' advertisement is the central issue of computational advertising and is subject to a complex set of competing objectives. *Advertisers* wish to maximise their return given a range of advertising spending. *Publishers* desire maximum revenue per impression in the short term. *Users* desire the most relevant ads. Finally, the *exchange* itself desires maximal long-term revenue.

This thesis considers a specific exchange in the German market. This exchange meets the objectives above by selecting an advertisement through an auction process: for each ad request from a publisher, the exchange retrieves all ads that meet constraints for the publisher and the user, calculates the per-impression value paid for each ad, and selects the highest-paying advertisement. For a cost-per-mille (CPM) advertisement, the value paid by the advertisement in the auction is simply the CPM. For performance-based advertisements, the exchange calculates the expected value by multiplying the value paid if an interaction occurs with the estimated probability of interaction. If this probability is consistently over-estimated for any given ad, then the exchange will rank this ad above other ads paying more per impression. If this probability is consistently under-estimated for any given ad, then the exchange will rank this ad below other ads paying less per impression. In both cases there is a loss in revenue that would otherwise have been captured. Estimating this interaction probability has motivated significant work in industry and academia due to its difficulty and economic relevance.

In the specific exchange considered, making a more accurate interaction probability estimation for CPA campaigns will change the outcome of many auctions, assigning different ads to user impressions, which may result in a higher average value paid per ad impression, higher revenue for the exchange and publisher, and higher return on investment for the advertiser (if their pricing is accurate).

As can be expected, the production implementation in the exchange is more complex than described. Advertisements are organised into multiple tiers according to more criteria than value paid, such as the advertiser's importance as a client, the ad category, and the publishers' preferences. In addition, publishers may specify a minimum price required for an ad to be shown. Where not specifically mentioned, these issues are ignored in the context of this thesis without loss of generality.

1.3 Real-time bidding

A recent development in the online advertising marketplace has been the advent of real-time bidding (RTB). In real-time bidding, each ad impression is auctioned off to the highest bidding advertiser or buy-side firm acting on an advertiser's behalf. An extension of the exchange auction above, this RTB auction process also occurs during the time a single web page loads in response to a user request (in less than 100 milliseconds). When a publisher requests an ad (through its own integrated technology platform, from an exchange, or from a supply-side firm), a 'bid request' is forwarded to external advertisers and buy-side firms, who each bid for the right to display an advertisement to the user concerned. In the context of an exchange, the ad selected from the exchange's internal auction is sent to the RTB auction, from which the final ad is selected.

A firm (platform) participating in the RTB market must respond to a bid request in less than 100 milliseconds in order to compete. Including intra-datacenter network latency, this leaves around 50 milliseconds to retrieve any relevant information (such as the user profile) for a bid request from a datastore, score one or more models, compute a bid, and respond to the request. At the time of writing, the largest participants may receive up to a million bid requests per second corresponding to hundreds of millions of users, placing significant throughput and latency requirements on the bid system. In addition, the largest firms manage Petabytes of data

1.4 Buy-side and sell-side platforms

An advertiser's goal is to maximise their return on investment by achieving the most user engagement possible for a given budget. The real-time bidding market provides access to millions of publisher websites and users, many more than the websites and users available through one or more exchanges. In addition, advertisers and marketing firms have the ability to directly select individual impressions likely to be more valuable (as they involve receptive users, an ideal time of day, context, or other reason). Purchasing individual impressions is impractical for most advertisers due to the complexity of the infrastructure and models required. Buy-side or demand-side platforms (DSPs) have evolved to purchase user impressions on behalf of advertisers from both premium inventory or direct-buy markets and non-guaranteed inventory from the real-time bidding market. By using a DSP, advertisers are only required to make broad decisions about the audience they wish to advertise to and the price they are willing to pay. The DSP tracks users, campaign performance, selects users likely to interact with the advertiser's campaigns, and purchases ad impressions for these users for the best possible price through the real-time bidding market [89]. The DSP selects the best impressions for a campaign by tracking and profiling users and considering the context and time of the impression, while competing with other advertisers and DSPs.

A publisher's goal is to maximise their return on each page view or impression. Publishers may contract with sell-side, or supply-side platforms, who work with publishers to sell ad impressions through the real-time bidding market at the highest prices possible. Sell-side platforms are beginning to conduct their own user and market profiling in order to gain the best possible revenue from each user impression.

The RTB market can be seen as a successor to and extension of the function of an exchange, selecting the 'best' ad from all participants in the online advertising market at a given instant rather than merely the campaigns and publishers managed by one exchange. Exchanges still operate in the RTB market, although as a participant. Supply-side platforms, DSPs, and exchanges have some similarities and some differences. A DSP's objective is to identify the best user impressions from the market for their client's marketing campaigns, and decide what to bid for these impressions. A DSP can watch a stream of bid requests and bid only when their models indicate that it will be profitable to do so. A supply-side platform desires to maximise the bids for each impression. Finally, an exchange must provide an advertisement for each integrated publisher request and therefore needs to accurately calculate the value of every user impression for each managed campaign, rather than identify and bid on the best impressions alone (often only 1-3% of total impressions). We note that firms may combine one or all of these functions, and that the complex RTB market has grown and fragmented between 2010-2014. Very large and small firms now provide a broad range of services.

1.5 Outline and Contributions

For both DSPs and exchanges, predicting the probability of user-ad interaction given a time and context in order to determine which ad to display or the price to bid for an impression is central to the companies' revenue. This project is undertaken within a large European ad exchange, referred to as 'the exchange' hereafter. The desired outcome from this project is to develop and validate a system for interaction probability prediction that improves on the exchange's existing estimation method, is useful in both the exchange and real-time bidding contexts, and is capable of reliably

processing hundreds of millions of events on a daily basis. The work described in this thesis therefore focuses on the application of existing (although recent) theory to this problem.

In chapter 2 we describe the existing method of estimating interaction probabilities based on aggregate historical data and provide an overview of the information available. In chapter 3 we propose using combination of a main effects model and a beta-binomial model which is conceptually similar to the existing method, can be implemented with little additional computational resources. In chapter 4 we investigate the effect of adding model parameters describing interactions between ads and the context in which they are displayed to the main effects model, resulting in a marginal improvement. Our contribution in these chapters consists of implementing a system which achieves a negative log-likelihood value over 10% less than the existing method measured over the period of a week.

In chapter 5, we model the user-ad interaction probability as a function of information specific to each impression event using a regularised generalised linear model (rather than using aggregate historical rates). We first develop a data pipeline based on the Hadoop software for processing the exchange event logs, and then use regularised generalised linear models to estimate the interaction probability for each impression event for each campaign. Our contributions in this chapter are primarily the implementation details such as the data pre-processing, feature engineering and sub-sampling methods, and the development of a stand-alone, scalable implementation capable of training thousands of campaign models over hundred of millions of impressions on a daily cycle using the Hadoop software ecosystem and Amazon Web Services.

At the time of development, the systems described in chapter 5 and chapter 3 provided more accurate interaction probability estimations than the method used in the exchange/demand-side platform with which this project was undertaken. We believe that these implementations would be suitable for commercial use with little modification. However, when this project was nearly complete, integrating the systems developed in this project into the ad exchange was rendered unnecessary by unforeseen and unrelated commercial circumstances. Therefore, despite good offline results these systems were not tested in production.

Consequently, in chapter 6, we investigate a number of common algorithms using the datasets developed in the previous chapters for personal interest and experience with high-performance computational techniques, not for their relevance to the industrial problem. We investigate generating additional user features with Laplacian eigenmap embedding, including investigating sub-quadratic algorithms for k-nearest neighbour calculation. We implement the recently developed FEAST eigensolver algorithm [105] in MATLAB, implement a simulated annealing algorithm for graph community detection via modularity maximisation, and investigate fitting stochastic block models to the bipartite user-website network. We also implement a linear manifold regularization method.

Finally, in chapter 7 we investigate the real-time bidding market in Germany between September and November 2013. We present selected results from this investigation, including how tracking and responding to market microstructure may provide a commercial advantage to an exchange or demand-side platform. We also propose using real-time bidding responses from competing firms as sources of additional information on users tracked by the exchange, improving the internal exchange models.

Chapter 2

Estimating user-ad interaction probability: the existing baseline and a road map

This chapter gives a high-level overview of how a medium-sized European ad exchange responds to publisher ad requests. First, we define some additional terms specific to the exchange considered. Second, we describe the ad auction which the exchange uses to select the best ad for a request, and explain the importance of estimating the probabilty that the user will interact with each candidate advertisement. We then describe the method currently used by the exchange for estimating this probability and its limitations.

2.1 The exchange auction, a local greedy strategy

Each publisher integrated with the exchange owns a number of *websites* composed of pages defined by a unique URL. The publisher may create one or more advertisement containers on each page, called *slots*. When a user's browser begins a request for a page, the publisher sends an *ad request* to the exchange for each slot on the page. This ad request contains the user and slot information. Let $\mathcal{U}$, $\mathcal{S}$, $\mathcal{A}$ denote the set of all users, slots, and ads tracked by the exchange, and let $u_h \in \mathcal{U}$, $s_j \in \mathcal{S}$, and $a_k \in \mathcal{A}$ denote elements of these sets. Then an ad request received at time t is of the form

$$\mathsf{ad\ request} \coloneqq \{u_h, s_j, t\} \tag{2.1}$$

The exchange needs to respond to the ad request with an ad that best satisfies the publisher, advertiser, and exchange objectives described in subsection 1.2.2. The exchange responds with the ad a_k it expects to have the highest *impression revenue*, the fee paid by the advertiser if the ad is displayed to the user. When the exchange responds to the ad request with a specific ad a_k, it is displayed to the user in an *impression* event

$$\mathsf{impression} \coloneqq \{u_h, s_j, a_k, t\} \tag{2.2}$$

The impression revenue for a CPM ad is simply the CPM value. The true impression revenue for a CPA ad is only determined after one ad had been chosen and displayed to the user in an impression event. If the user interacts with the ad, an interaction event takes place, and the impression revenue becomes the CPA value. If the user does not interact with the ad, then the impression revenue is zero. In order to compare different types of ads during the selection process, the *expected impression revenue* (eCPM) is used.

For each impression event an interaction event occurs or does not occur, which can be considered a Bernoulli trial $\bar{Y}_{hjk} \in \{0, 1\}$. The probability of success is dependent on the user, slot, ad, and time of the impression, denoted by h, j, k, t respectively. Stated formally

$$\bar{Y}_{hjkt} \sim \text{Bernoulli}(p_{hjkt}) \tag{2.3}$$

$$p_{hjkt} = \Pr(\bar{Y}_{hjkt} = 1 | u_h, s_j, a_k, t) \tag{2.4}$$

The expected impression revenue for a CPA ad is $\Pr(\bar{Y}_{hjkt} = 1 | u_h, s_j, a_k, t) \times \text{CPA value}(a_k)$.

The selection process for the best ad $a^* \in \mathcal{A}$ for a specific ad request $\{u_h, s_j, t\}$ can then be written as

$$a^* \mid \{u_h, s_j, t\} = \underset{a_k \in \mathcal{A}}{\text{argmax}} \begin{cases} p_{hjkt} \times \text{CPA value}(a_k) & \text{if } a_k \text{ is a CPA ad} \\ \text{CPM value}(a_k), & \text{if } a_k \text{ is a CPM ad} \end{cases} \tag{2.5}$$

where CPA value(a_k) is the value paid if the user interacts with a CPA ad a_k and CPM value(a_k) is the known value paid for displaying the CPM ad a_k. The value $p_{hjkt} = \Pr(Y_{hjkt} = 1 \,|\, u_h, s_j, a_k, t)$ is the unknown probability of the user interacting with the ad in that context at that time. Estimating this value is therefore a significant part of selecting the best advertisement.

2.2 The exchange's existing probability estimation method

The method for estimating $\Pr(\bar{Y}_{hjkt} = 1 \,|\, u_h, s_j, a_k, t)$ currently implemented in the exchange is described next.

2.2.1 The basic model

The number of interaction events (successes) given a number of impression events (trials) can be modelled as a Binomial random variable Y_{hjkt} with parameter p_{hjkt}. The exchange's existing method of calculating the interaction probability makes the assumption that the probability is independent of the user and time, and only dependent on the ad a_k and slot s_j, written as follows

$$\Pr(Y_{hjkt} = 1 \,|\, u_h, s_j, a_k, t) \approx \Pr(Y_{jk} = 1 | s_j, a_k) \tag{2.6}$$

The number of interactions (successes) S_{jk} occurring for a given number of impression events (trials) T_{jk} conditioned on the slot s_j and ad a_k is then given by the modified Binomial random variable

$$Y_{jk} \sim \text{Binomial}(T_{jk}, p_{jk}) \tag{2.7}$$

$$p_{jk} = \Pr(Y_{jk} = 1 | s_j, a_k) \tag{2.8}$$

The estimate of $\Pr(Y_{hjkt} = 1 | u_h, s_j, a_k, t)$ used by the exchange is the maximum likelihood estimate (MLE).

$$\hat{p}_{jk} = \begin{cases} \frac{S_{jk}}{T_{jk}} & \text{if } T_{jk} > 0 \\ \text{unknown} & \text{if } T_{jk} = 0 \end{cases} \tag{2.9}$$

However, for the majority of ad-slot pairs s_j, a_k a statistically significant number of impression events does not exist.[1] In order to solve this problem, the exchange combines impressions from multiple ad-slot pairs $\{s_j, a_k\}$ in order to create a large enough sample. For any subset of ads $\mathcal{X} \subseteq \mathcal{A}$ and subset of slots $\mathcal{Z} \subseteq \mathcal{S}$ the number of impression events $\{u_h, s_j, a_k, t\} \,|\, s_j \in \mathcal{Z},\, a_k \in \mathcal{X}$ is given by $T_{\mathcal{X}\mathcal{Z}}$. The number of interaction events is given by $S_{\mathcal{X}\mathcal{Z}}$. The maximum likelihood estimator $\hat{p}_{jk}$ from (2.6) for an ad and slot can then be replaced by the maximum likelihood estimate for the sets $\mathcal{X}$ and $\mathcal{Z}$

$$\hat{p}_{jk} \approx \hat{p}_{\mathcal{X}\mathcal{Z}} = \begin{cases} \frac{S_{\mathcal{X}\mathcal{Z}}}{T_{\mathcal{X}\mathcal{Z}}} & \text{if } T_{\mathcal{X}\mathcal{Z}} > 0 \\ \text{unknown} & \text{if } T_{\mathcal{X}\mathcal{Z}} = 0 \end{cases} \tag{2.10}$$

assuming $a_k \in \mathcal{X}$ and $s_j \in \mathcal{Z}$. For large enough sets $\mathcal{X}$ and $\mathcal{Z}$, a statistically significant sample required for the maximum likelihood estimate to be reliable will exist. We now consider how to select additional ad-slot pairs to form the sets $\mathcal{X}$ and $\mathcal{Z}$ given the ad-slot pair $\{s_j, a_k\}$ considered in the estimation of p_{jk}.

2.2.2 Data hierarchies

Ads and slots can be considered to belong to a hierarchical structure. For example, ads can be considered to belong to advertising campaigns. In turn, campaigns belong to advertisers, the top level of the ad hierarchy. An example is a graphical or video ad for an airline flight package which is part of a campaign run by a major airline (the advertiser). Slots are associated with a nominal (integer) website identifier. Websites in turn belong to publishers, the top level of the slot hierarchy. For example, a slot may be placed on a news article page, which is part of a newspaper website, which is owned by a publishing group. There is a global level consisting of all ads and slots. Due to the hierarchy, if an ad belongs to a campaign, and the campaign to a publisher, then the ad belongs to the publisher. It is natural to assume that the contribution to p_{jk} from ads within the same campaign may be correlated due to shared properties, and to a lesser extent for ads within the same advertiser.

Recall that $\mathcal{A}$ is the set of all ads. Let $\mathcal{A}_c$ be the set of ads associated with a particular campaign, i.e. $\mathcal{A}_c \subseteq \mathcal{A}$. Similarly let $\mathcal{A}_a$ be the set of ads belonging to a particular advertiser, i.e. $a_k \subseteq \mathcal{A}_a$. On the slot side, let $\mathcal{S}_w$ be the set of slots associated with a particular website, i.e. $s_j \subseteq \mathcal{S}_w$. Similarly let $\mathcal{S}_p$ be the set of slots belonging to a particular publisher, i.e. $s_j \subseteq \mathcal{S}_p$. For convenience, let $\mathcal{C}$ be the set of all $\mathcal{A}_c$, $\mathcal{A}'$ be the set of all $\mathcal{A}_p$, $\mathcal{W}$ be the set of all $\mathcal{S}_w$, and $\mathcal{P}$ be the set of all $\mathcal{S}_p$. Then possible sets $\mathcal{X}$ and $\mathcal{Z}$ are given by

$$\mathcal{X} \subseteq \mathcal{A} \cup \mathcal{C} \cup \mathcal{A}' \tag{2.11}$$

$$\mathcal{Z} \subseteq \mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \tag{2.12}$$

For an incoming ad request, the exchange first checks if a large enough number of impressions has been observed for the ad, slot pair. If not, the exchange selects a set $\mathcal{X}$ and $\mathcal{Z}$ containing more ad-slot pairs and checks if a large enough number of impressions has been observed for these pairs. If not, the exchange selects larger sets $\mathcal{X}$ and $\mathcal{Z}$ and so on.

[1] A typical interaction rate is around 1 in 1500, which requires thousands of impression events to accurately estimate p_{jk}.

The specific choices of $\mathcal{X}$ and $\mathcal{Z}$ at each step are given in the following table from the ad-slot pair at the most specific level to the most general level of the set of all ads and the set of all slots.

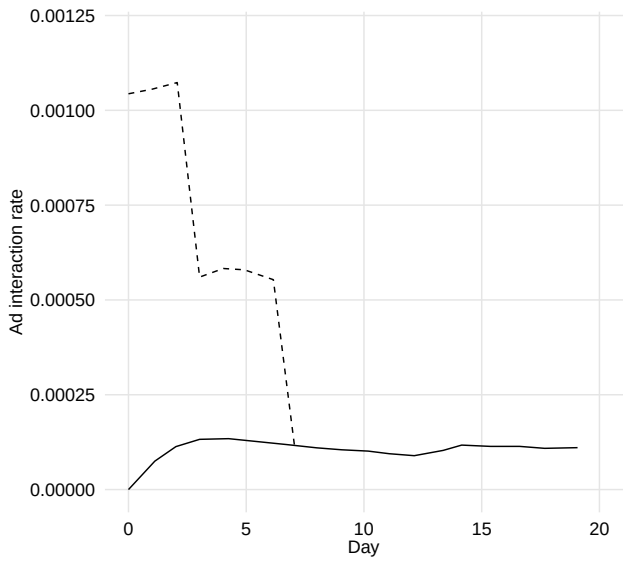
	$\mathcal{X}$	$\mathcal{Z}$	minimum impressions
most specific	a_k	s_j	n_1
	a_k	S	n_2
	$\mathcal{A}_c$	S	n_3
	$\mathcal{A}_a$	S	n_4
most general	$\mathcal{A}$	S	

Table 2.1: Hierarchical aggregation levels used for maximum likelihood interaction rate estimate

The values n_1,n_2, n_3, and n_4 used in the exchange are constant and were set heuristically at some point in the past. The hierarchical entities exist as a result of the entities used in the exchange's commercial database. We note that a slot may be displayed on multiple webpages and/or websites and be shifted between websites, or switched on or off, without notice. This makes the parent-child relationships between the entities in the slot hierarchy less significant than they otherwise could be. We also note that it would be natural to define a webpage by a unique URL instead of an integer website identifier. However, the exchange data collection did not support this until late in the project, an outcome of this work.

2.2.3 Example of the existing method

Figure 2.1 shows this stepped estimation process. The solid line shows the maximum likelihood estimate at the ad-slot level of the hierarchy, plotted over time. The dotted line shows the hierarchical exchange estimation, where the steps shown correspond to the campaign, ad, and finally ad-slot levels of the hierarchy. The final level is reached on the 7th day after the start of the chart.



(a) Ad-slot pair example

Figure 2.1: Example of the stepped hierarchical exchange estimation process.

In Figure 2.1 the hierarchical maximum likelihood estimate process does not reach the ad-slot level until 7-8 days after the ad is first shown, and before this point, estimates an interaction rate much larger than the empirical long-term rate for the ad-slot pair. The maximum likelihood estimate at the ad-slot level alone stabilises at the true interaction rate after 2-3 days. Quick convergence to a good estimate of the true interaction rate is important for the objectives described in subsection 1.2.2, as in this example the ad is being mis-priced during the first 7-8 days.

2.3 Drawbacks of the existing exchange method for interaction probability estimation

While the global average interaction probability for an impression event is around 7×10^{-4}, variations of an order of magnitude in interaction probability between ad-slot pairs with the same ad or slot in the pair are common. The distribution of ad-slot interaction probabilities is generally similar to an exponential distribution but with a heavier tail. If the true interaction probability for an ad-slot pair is equal to the global mean, the sample size required in order for there to be a 95% probability that the maximum likelihood estimate of the interaction probability using the ad-slot pair events is at least 80% of the true value is $9,396$. For a true probability of 1×10^{-4} the sample size required rises to $65,789$, which is approximately the value of $\mathcal{A}$ in Table 2.1.

Therefore, constant values for n_1, n_2, n_3, and n_4 (the number of impressions required to transition between the global advertiser, campaign, ad, and ad-slot levels in the existing method) are sub-optimal for a significant fraction of advertisements. To illustrate, the majority of transitions between these hierarchical estimation methods change the estimated interaction probability by a factor of 2 or more, and in many transitions it changes by a factor of 10 or more.

In the existing system, the immediate context for an impression event (the slot) is not considered in the hierarchical group maximum likelihood estimate until the ad-slot level of the stepped process has been reached, by which time a significant fraction of the advertisement's budget has typically been spent. There is also a wide variation in interaction rates between slots for a single website and publisher, influencing the ad performance. In addition, the Binomial random variable $\bar{Y}_{jk}$ and maximum likelihood estimate used to model the interaction probability assumes that the interaction rate does not change with time. However, user impression events correspond to a non-stationary Bernoulli process with an interaction rate that varies significantly with time. Finally, the existing method ignores the user component of the interaction probability, as methods that model the user component require more advanced data pipelines and modelling approaches than the maximum likelihood estimate discussed above. There is also a large variation in mean interaction probability between ads (or slots), a statement which is also true for higher-up entities in the ad and slot hierarchies.

2.4 Online and offline evaluation criteria for proposed models

The ideal evaluation criteria for any interaction probability estimation method developed in this project is that it generates more revenue than the existing method, all other factors being equal. The only definitive way of testing for this criteria is to run each system in parallel and direct ad requests to each system at random. However, in order to evaluate candidate methods offline, a proxy for this evaluation method is required that can be used in isolation with historical data. Developing

an accurate and precise proxy is difficult, as small changes to the predicted interaction probabilities at time t affect future auction outcomes. We use the the binomial log-likelihood calculated over all impression events as a proxy. The log-likelihood is defined by

$$l(\boldsymbol{\phi}) = \sum_{i=1}^{N} \left(y_i \log(f(\boldsymbol{\phi}; x_i)) + (1 - y_i) \log(1 - f(\boldsymbol{\phi}; x_i)) \right) \tag{2.13}$$

where N is the number of test events, $y_i \in \{0, 1\}$ is the ocurrence or non-occurrence of an interaction for an impression event, and $f(\phi)$ is the prediction function for the interaction probability.

We evaluate proposed and existing models by training each model on a historical dataset from the exchange and then comparing the log-likelihood for the interaction probabilities generated by each model for held-out test data from after the training period. We note that using the log-likelihood for evaluating the performance of an interaction probability prediction system in the context of an exchange or demand-side platform suffers from a number of shortcomings. Most significantly, the binomial log-likelihood penalises large errors to a greater extent than small errors, on a continuous scale. However, the outcome of an ad auction is a discrete event, and a small error which changes the auction ranking has a greater impact than a large error which does not change the ranking and auction outcome. However, developing a more application-specific method is more time-consuming than simply testing the new method on a small fraction of live traffic.

We also note that in the situation where a model is used to predict interaction probabilities for an ad-slot pair the root-mean-square error (RMSE) between predicted and actual interaction probability is a very poor error metric. It assumes that the errors are all from a normal distribution, which is a terrible assumption for count data; Poisson is a reasonable assumption. The RMSE also does not account for the number of impressions for each ad-slot pair, unlike the log-likelihood. It is dominated by low-confidence outliers in a similar fashion to ordinary linear regression. The RMSE also heavily penalises large errors while minimally penalizing small errors (although the log likelihood does this also). Given the distribution of interaction probabilities in the ad auction it may be more beneficial to overall revenue to place a higher penalty on the large number of smaller mispredictions near the mean value than the occasional large misprediction.

2.5 Information available to the exchange and demand-side platform

2.5.1 General context

We first consider information that may be available to a demand side platform at the time an ad request or a real-time bid request is received in general terms. This information is created from the exchange publisher and ad databases and the historical event log, and is typically related to

- the candidate advertisements
- the webpage, slot, app, or other direct context in which the advertisement is to be displayed
- further contextual information such as the time of day, weather, external events, or device
- the user interacting with the online content

Information about the specific webpage or slot in the ad request may include the publisher or brand to which the webpage belongs, the content and spatial layout of the page, and the position of the advertisement container within the layout. At the time of writing, revenue attribution models are beginning to take these features into account when deciding whether to attribute a purchase to online ads previously displayed to a user [102].

Information about candidate advertisements may include the brand, type of product or service, and semantic features generated from the advertisement itself. Further information can be found in [77], [37], and [58].

Information about the context of the ad request, or external factors that may affect the probability of an interaction, include (for example) the time of day, the day of the week, and many other factors such as the weather conditions. Contextual information can also include the influence of external events such as the start of the school year, and bank holidays.

Information available about the user may include the user's IP address and browser version alone, or demographic information and a detailed history of browsing behvaiour, webpages visited, previous ad interactions, location through smartphone tracking, and previous purchases. This information is used to build a profile of the user which provides insight into likely future purchasing behvaiour.

2.5.2 Specific context

Very little of the information described in subsection 2.5.1 is available in the context of this project. Advertisements, slots, websites, advertisers, and publishers are only identified by unique integer identifiers, and no semantic, categorical, brand, or other such useful information is avilable. We also note that the exchange or demand-side platform's decision engine must run in under 50 milliseconds and ideally in under 10, including evaluating any mathematical models or sets of decision rules, which makes collecting this information during the decision-making process infeasible. We note that actions such as scraping each publisher webpage ahead of time and engineering and validating semantic features (in German) is of great interest but beyond the scope of this project. This is the subject of ongoing research at the time of writing.

We now describe the information that is available in the context of this project. The information is entirely derived from the event logs described in the section above. For every event that occurs in the exchange or demand-side platform system, a line of text describing the event is written to one of many server log files. Events include (but are not limited to) an ad being displayed to a user on a client's site in response to an ad request, a user interacting with an ad, a bid request being sent to real-time bidding partners, bid responses being received from real-time bidding partners, and tracking pixels being viewed. An event log line is similar to the following representation

```
impression_id timestamp ad_id slot_id user_id event_type price bid
```

Not all fields are relevant to every event. The ad and slot can be used to identify the corresponding website, publisher, campagin, and advertiser.

2.6 Selecting appropriate algorithms to improve the existing method

The machine learning algorithms appropriate to a computational advertising problem depend on the context, such as sponsored search or ad targeting, the intended outcome such as ranking, probability estimation, or user segmentation, and the information available, such as historical interaction rates only or a significant quantity of external features. Another important factor is the size of the datasets under consideration. Desired outcomes for this project include estimating rates of rare events, ad interaction probabilities, and user segmentation. We now briefly and non-exhaustively mention some families of algorithms applicable to computational advertising.

- Historical events and hierarchical entity structure can be used with many canonical and specific methods in the context of computational advertising. As an example, a sophisticated custom method for this context is given in [3]. We consider Bayesian hierarchical models and a main effects model as a way of accomplishing this in chapter 3.

- While descriptive features or external information can be utilised by a number of methods, regression is among the most significant. We consider regularised generalised linear models in chapter 5. Linear models of this type are highly scalable, easy to interpret, and can be used for feature selection. Linear model formulations can be differentiated by the choice of features and the model target and target distribution.

- Unsupervised learning describes finding patterns in data without using labelled examples. In chapter 6 we describe clustering internet users using their browser history, and clustering a bipartite graph formed from user-website interactions, both examples of unsupervised learning. Dimensionality reduction is often used as a pre-processing step prior to applying a model such as a linear model, and falls under unsupervised learning. Dimensionality reduction algorithms include principal component analysis, singular value decomposition, stochastic neighbour embedding, and locally linear embedding.

- Latent factor models include a class of generative models that explain observed interactions between entities in terms of latent properties of the entities, also referred to as collaborative filtering models. Inferring these latent properties using observed interactions allows the prediction of unobserved interactions. No explicit descriptive information on entities is required. We investigate latent factor models in chapter 4.

- We do not consider using neural networks due to lack of time.

- The Naive Bayes algorithm and algorithms based on decision trees or random forests have been found to increase the predictive accuracy for this dataset, and are the subject of ongoing research by others. Therefore, these methods are not considered in this project. In practice, the results of these models could be combined with the models described in this thesis.

Summary

This chapter gives an overview of the existing exchange functions and describes the ad auction which is at the heart of the ad delivery process. The method currently used by the exchange

for estimating the advertisement value and user interaction probability is presented, along with a discussion of methods for evaluating proposed models intended to superseede the existing method.

Chapter 3

Extending the existing method with hierarchical and main effects models

The existing interaction probability estimation method described in chapter 2 suffers from shortcomings described in section 2.3. However, it can be seen as a basic heuristic approximating a heirachical model which makes a smooth transition from a prior belief based on hierarchical averages to the observed interaction probability for a specific pair $\{a_k, s_j\}$. In this chapter we implement a gamma-Poisson model for individual ad-slot pairs that accounts for temporal drift in interaction probability and compare this model to the existing method. However, this has high computational cost. We then model the interaction probability for an ad-slot pair as a combination of main effects from the entities in the ad-slot hierarchies. This main effects model is used to create a beta prior for a beta-binomial distribution describing each ad-slot pair, which is updated as events for each specific ad-slot pair are observed. We continue to use the concepts and notation from chapter 2.

3.1 Related work

Bayesian hierarchical models and multi-level regression models are extremely common in many branches of scientific literature and are well covered in common texts such as [48], [70], and [49]. For the specific problem of predicting interaction probabilities given ads, slots, ad and slot hierarchies, and external covariates, a few works have been published by researchers working with online advertising firms. Two of the best of these are described below.

Lee et al. [75] models aggregate interaction probabilities at chosen levels of the ad and slot hierarchicies using separate binomial distributions, and forms an estimate of the specific ad-slot interaction probability by combining the parameters of these distributions with logistic regression models. The motivation for this work is improving the methods used at `Turn.com`, a global demand-side platform. Another notable approach is proposed by Agarwal et al. [3]. The authors estimate the interaction rate for each ad-slot pair by combining a baseline probability obtained from covariates and a logistic regression model with a multiplicative factor for each entity in the (un-ordered) Cartesian cross product of the ad and slot hierarchies. Suitable priors are placed on these factors, which reduces the number of parameters required and increases the computational efficiency. The motivation for this work is improving the methods used at `Yahoo.com`, a global technology, advertising, and content company.

These works assume that interaction rates for children of the same parent are correlated. For example, interaction probabilities for ads related to the same campaign, or slots placed on the same webpage are expected to be correlated.

3.2 Using Bayesian hierarchical models

3.2.1 Beta-binomial hierarchical models

Recall from section 2.1 that the ad request is of the form of $\textsf{ad request} = \{u_h, s_j, t\}$. For each ad, we desire to estimate the probability $p_{hjkt} = \Pr(Y_{hjkt} = 1 | u_h, s_j, a_k, t)$ of an interaction, should the ad a_k be sent in response to the ad request. In this chapter, we also make the assumption that the probability is independent of the user as follows

$$\Pr(Y_{hjkt} = 1 \,|\, u_h, s_j, a_k, t) \approx \Pr(Y_{jkt} = 1 | s_j, a_k, t) \tag{3.1}$$

This is done as there are up to 10^5 slots and advertisements at any point in time, but up to 10^8 unique users which significantly increases the difficulty of the problem. Note that we remove this assumption and explicitly consider the user contribution along with other event-specific features in chapter 5.

The existing exchange method models the number of interaction events, or successes S_{jk} for an ad-slot pair given a number of impression events T_{jk} as a Binomial random variable and uses the maximum likelihood esimtate for p_{jk}. As the majority of ad-slot pairs $\{a_k, s_j\}$ do not have a statistically significant number of impression events, the $\{a_k, s_j\}$ pair's maximum likelihood estimate is replaced with a maximum likelihood estimate calculated using groups of $\{a_k, s_j\}$ pairs selected using the entity hierarchy. As discussed in section 2.3, problems with this approach include the high variation in ad-slot pair interaction probabilities within the hierarchical groups and the longer-than-necessary period before the individual ad-slot pair's maximum likelihood estimate is used.

We desire to use the information avaialble in the events for the particular $\{a_k, s_j\}$ pair as soon as they are observed. A useful model in this situation is a Bayesian beta-binomial hierarchical model. This model blends the estimate for a single ad-slot pair with an estimate obtained using a group of ad-slot pairs obtained from the hierarchy (differently to the existing method), given by all $\{a_k, s_j\}$ pairs where $a_k \in \mathcal{X}$ and $s_j \in \mathcal{Z}$. One such model is required for each ad-slot pair.

The interaction probability p_{jk} for each ad-slot pair is assumed to be drawn from a Beta distribution as $p_{jk} \sim \text{Beta}(\pi, M)$ with mean $\pi = \alpha/(\alpha + \beta)$ and shape parameter $M = \alpha + \beta$ which results in an ad-slot interaction rate variance of $\sigma^2 = \pi(1 - \pi)/(1 + M)$. The maximum likelihood esimate for an individual $\{a_k, s_j\}$ pair is given by $\hat{p}_{jk} = S_{jk}/T_{jk}$ and the Beta-binomial shrunken estimate $\hat{p}^s_{jk}$ is given by $w_{jk}\pi + (1 - w_{jk})\hat{p}_{jk}$, where the shrinkage factor w_{jk} is defined

$$w_{jk} = \frac{\pi(1 - \pi)/\sigma^2 - 1}{(\pi(1 - \pi)/\sigma^2 - 1) + T_{jk}}.$$

A hyper-prior for α and β resulting in a proper posteriors can be constructed using the parameterization $(\log(\alpha/\beta), \log(\alpha + \beta))$ as

$$\Pr\left(\log(\alpha/\beta), \log(\alpha + \beta)\right) \;\propto\; \alpha\beta(\alpha + \beta)^{-5/2}$$

Methods for computing the maximum likelihood estimates of the parameters of this Beta-Binomial model are well studied in the literature [53] and available in many computational statistics libraries [104].

3.2.2 Gamma-Poisson hierarchical models

The model above does not take temporal variation in the $\{a_k, s_j\}$ pair interaction probability into account; this variation is clearly evident in Figure 3.1. In order to account for temporal variation we choose the basic method of considering more recent events to be more representative of the current interaction probability than events further in the past. This is formalised by applying exponential smoothing to the numbers of interactions and impression events for each $\{a_k, s_j\}$ pair over discrete time bins according to

$$S_{jk}^{(t)} \coloneqq \gamma S_{jk}^{(t)} + (1-\gamma) S_{jk}^{(t-1)} \tag{3.2}$$

$$T_{jk}^{(t)} \coloneqq \gamma T_{jk}^{(t)} + (1-\gamma) T_{jk}^{(t-1)} \tag{3.3}$$

where $S_{jk}^{(t)}$ and $T_{jk}^{(t)}$ are the number of interaction and impression events, respectively, for an $\{a_k, s_j\}$ pair at time bin t, taken as 24 hours to avoid complications due to intra-day cyclical variation. However, this does not result in integer counts for the number of interactions and impression events, which are no longer directly represented by a binomial random variable.

We therefore consider a gamma-Poisson hierarchical model, as the low rate of occurrence of interaction events allows the expected number of successes ocurring for a given number of impression events to be approximated by a Poisson distribution. In the gamma-Poisson model, the expected number of interaction events $S_{jk}^{(t)}$ given $T_{jk}^{(t)}$ impression events for an $\{a_k, s_j\}$ pair at time t is considered to be drawn from a Poisson distribution

$$g\left(S_{jk}^{(t)} | \lambda_{jk}^{(t)}\right) = \text{Poisson}(\lambda_{jk}^{(t)} T_{jk}^{(t)}) \tag{3.4}$$

From here onwards we assume all calulations are performed with temporally adjusted counts at time t and drop the annotation $^{(t)}$ for notational clarity. The interaction rate λ_{jk} for a given pair $\{s_j, a_k\} \,|\, s_j \in \mathcal{Z},\, a_k \in \mathcal{X}$ is drawn from a Gamma distribution

$$g(\lambda_{jk} | \alpha, \mu) = \text{Gamma}\left(\alpha, \frac{\alpha}{\mu}\right) \tag{3.5}$$

Given this choice of parametrization, the mean of the gamma distribution is μ and the variance is given by μ^2/α. The value of α controls the variance in the interaction rate between $\{a_k, s_j\}$ pairs: $\text{Var}(\lambda)$ goes to 0 as $\alpha \to \infty$. The priors for α and μ can be specified as the gamma distribution

$$\pi(\alpha) = \text{Gamma}(s_0, s_1) \tag{3.6}$$

$$\pi(\mu) = \text{Gamma}(s_2, s_3) \tag{3.7}$$

where the s_i are hyperparameters selected arbitrarily prior to fitting the model. This distribution has a mode of zero and infinite mean and variance, a minimally informative prior. After solving for the posterior distribution of α and α/μ, the conditional posterior for λ_{jk} can be expressed (due to the choice of prior) as follows

$$g(\lambda_{jk} | data, \alpha, \mu) = \text{Gamma}(S_{jk} + \alpha, T_{jk} + \alpha/\mu) \tag{3.8}$$

The posterior mean for λ_{jk} at time bin t can also be written as a convex combination of the interaction rate for an $\{a_k, s_j\}$ pair and the pooled interaction rate as follows

$$E(\lambda_{jk} \,|\, data) = E(w_{jk} \,|\, data)\frac{S_{jk}^{(t)}}{T_{jk}^{(t)}} + (1 - E(w_{jk} \,|\, data))\frac{S_{\mathcal{X}}^{(t)}}{T_{\mathcal{Z}}^{(t)}} \tag{3.9}$$

where

$$w_{jk} = \frac{\alpha}{\alpha + T_{jk}^{(t)}\mu} \tag{3.10}$$

The extent to which $E(\lambda_{jk} \,|\, data)$ is shrunk towards the prior mean will decrease as T_{jk} increases or as the prior variance increases (an uninformative prior). In the first case this is because a large sample for an individual pair $\{a_k, s_j\}$ is more significant than the pooled estimate The shrinkage moves the interaction rate for $\{a_k, s_j\}$ pairs with a small number of observations towards the mean to a greater extent than it does for pairs with a large number of observations, meaning that with adequate implementation the interaction rate will never be estimated as zero (as it would be using the maximum likelihood estimate for the individual pair alone).

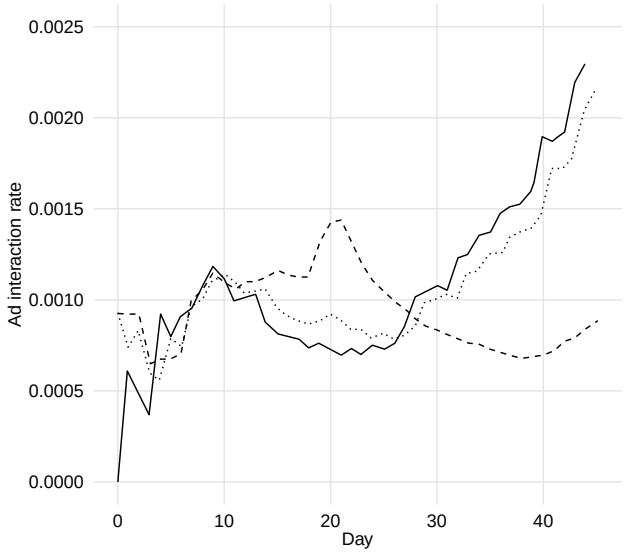
Posterior distributions for α, μ and λ_{jk} are found using MCMC sampling using the JAGS software package and the rjags R package, due to the non-standard posterior for α. The model for a given ad-slot pair is recalculated once sufficient additional data has been observed. We note that an exponential distribution $\text{Exp}(s_0, s_1)$ or the Lomax distribution could be used as prior distributions for α and μ, as suggested by Christiansen and Morris [36].

3.2.3 Gamma-Poisson computational experiments

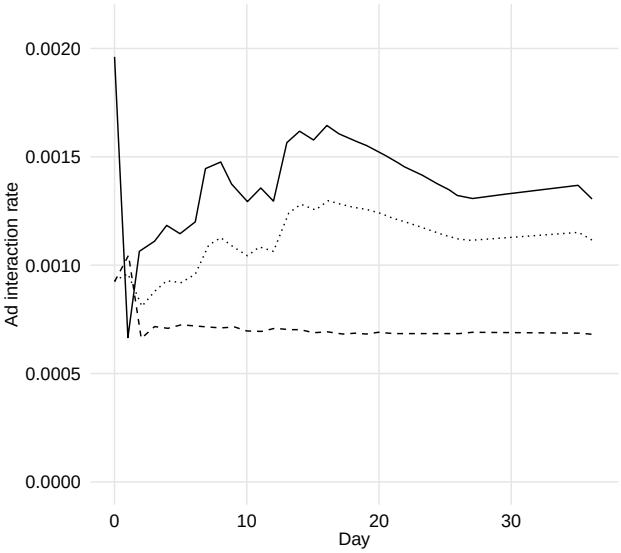
We evaluate the gamma-Poisson model above for three randomly selected ad-slot pairs to illustrate the performance of different methods. Figure 3.1a, Figure 3.1b, and Figure 3.1c show click-through rates over time for the existing method and the proposed method for each pair. The proposed method clearly outperforms the existing method at moving from the prior hierarchical belief towards the true click-through rate, and adapting to temporal variation.

In Figure 3.1c above, the transitions made by the existing method between the advertiser maximum likelihood estimate, the campaign maximum likelihood estimate, and the $\{a_k, s_j\}$ maximum likelihood estimate are clearly seen. In this case the campaign maximum likelihood estimate equals the ad maximum likelihood estimate as the campaign has one ad. The proposed method approaches the underlying (variable) ad-slot probability more quickly due to the greater weight placed on observed ad-slot events, eventually transitioning to the $\{a_k, s_j\}$ maximum likelihood estimate alone on the same day as the baseline method.

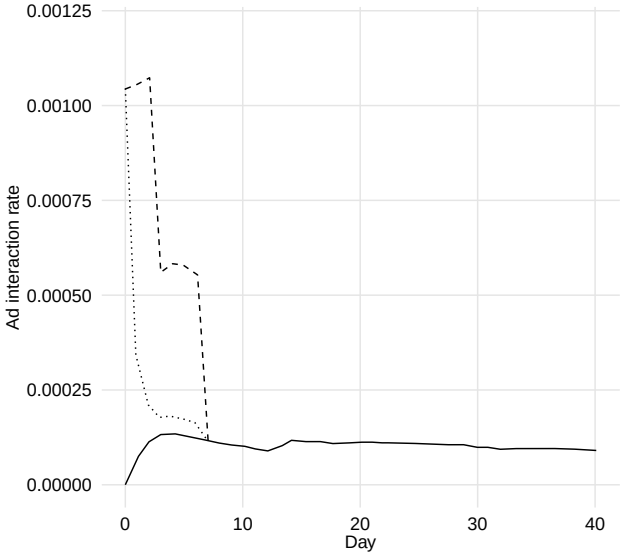
The gamma-Poisson model described above showed excellent performance and improved on the existing method for every ad-slot pair tested, approximately 100 randomly selected pairs. However, the computational requirements of this method are too high to make it a viable candidate for production use (as implemented here), as there may be hundreds of thousands of ad-slot pairs for which to evaluate a model. We therefore consider a computationally simpler approach.



(a) Ad-slot pair 1



(b) Ad-slot pair 2



(c) Ad-slot pair 3

Figure 3.1: Ad interaction rate plotted against time for 3 example ad-slot ($\{a_k, s_j\}$) pairs with a non-trivial amount of traffic. The horizontal axis displays the number of days since the ad first appeared on the slot. The vertical axis displays the ad interaction rate for the slot. The solid line is the true interaction rate. The dashed line is the exchange's existing estimation. The dotted line is the gamma-Poisson model.

3.3 Combining a main effects model with a beta-binomal distribution

We now model the interaction probability for each ad-slot pair as an additive combination of a constant global term and a constant bias term for each of the publisher, advertiser, campaign, website, ad and slot entities

$$\Pr(Y_{jk} = 1|s_j, a_k) = p_{jk} \coloneqq \sigma(\mu + b_p + b_a + b_c + b_w + b_k + b_j) \tag{3.11}$$

where Y_{jk} is the outcome of an impression event for the ad-slot pair $\{a_k, s_j\}$, $\sigma(\cdot)$ is the logistic function, μ is the global mean interaction probability, and $b_p, b_a, b_c, b_w, b_k, b_j$ are constant offsets for the publisher, advertiser, campaign, website, ad, and slot respectively. The bias terms are included in the model to compensate for the fact that many hierarchical ad and slot entities have vastly different observed interaction rates. For instance, an ad with a high interaction rate on average is likely to have a lower interaction rate than this value when displayed on a slot with a very low average interaction rate. The model parameters can be estimated with respect to the log-likelihood using the following optimization problem

$$\underset{b_p,b_a,b_c,b_w,b_j,b_k}{\operatorname{argmin}} \sum_{j,k} -T_{jk} \log p_{jk} \; - \; (T_{jk} - S_{jk}) \log(1 - p_{jk}) +$$
$$\lambda_1 \sum b_p^2 + \lambda_2 \sum b_a^2 + \lambda_3 \sum b_c^2 + \lambda_4 \sum b_w^2 + \lambda_5 \sum b_j^2 + \lambda_6 \sum b_k^2$$

Where p_{jk} is the observed interaction probability for the ad-slot pair $\{a_k, s_j\}$ and λ_1 and λ_2 are regularization hyperparameters shrinking the bias terms toward zero to avoid overfitting. This optimization problem could be solved using an L-BFGS implementation such as the FORTRAN library by Zhu, Byrd, and Nocedal [85], or stochastic or coordinate descent, or other methods. We demonstrate stochastic gradient descent by way of example. For each training ad-slot pair, we loop through the model parameters, updating each parameter in the opposite direction to the gradient, using the following general update rule

$$b_* \leftarrow b_* + \alpha_* \left(\frac{e^a T_{jk}}{1 - e^a} - S_{jk} \right) + 2\lambda_* b_*$$

where the α_* terms are parameter-specific learning rates and the λ_* terms are parameter-specific regularization terms. We loop through the training examples multiple times, until the error on a validation set (held out from the training set) starts increasing due to overfitting.

For each ad-slot pair, the output from this model is used to set the parameters of a Beta prior distribution for the ad-slot interaction probability. The variance of the Beta prior is set heuristically and optimised using held-out sets of training data. Every night, the optimization problem above is re-solved, and as additional impression events are available for the ad-slot pair, the posterior of each ad-slot Beta distribution is updated. When making predictions for an ad or a slot which does not appear in the training data, some of the relevant bias factors will not have been trained. In this situation they are set to zero.

3.4 Results and discussion

We train the model above on the full exchange dataset for the month of June 2013, only considering ad-slot pairs with more than 5000 impression events to prevent ad-slot pairs with only a few impression or interaction events from adversely affecting the training information. The model is tested on the full exchange dataset across all ad-slot pairs (no impression limit) for the first week of July 2013, consisting of tens of thousands of ad-slot pairs and hundreds of millions of impressions. The impression and interaction events from this test week are aggregated for each ad-slot pair, and the log-likelihood of the predictions made by each model are compared. The log-likelihoods are shown in Table 3.1. The proposed model offers a 19% improvement in log-likelihood over the existing method. An interesting and unexpected result is that the bias term model increases the RMSE over the existing exchange method while decreasing the log-likelihood, supporting the comments on the unsuitability of the RMSE as a performance metric in section 2.4.

Model	Log-likelihood	RMSE	AIC
existing exchange method	-1,890,565	1.213	3,689,293
main effects/beta model	-1,518,400	1.365	3,071,583

Table 3.1: Model performance

We chose not to re-evaluate the models every day of the test week, as would be done in production. This would be of greater benefit to the proposed model than the existing model, as the proposed model moves from the prior prediction to the observed value in a more effective manner for all slots.

Figure 3.1b illustrates that the proposed method estimates an ad interaction rate than can be a factor of 2 or more different from the existing method. Implementing a new model which changes the estimated interaction probability for CPC or CPA campaigns will result in many auctions having different outcomes than with the existing method. As more auctions take place, different campaigns wil be served to users than under the existing system. This changes the training data collected by the system, which has external effects such as changing the timestamps at which campaigns reach frequency caps. The methods proposed in this chapter will change the behaviour of the system in ways that are difficult to predict with historical data. Therefore, experimental validation of the impact of this model on revenue is essential before implementing it across all campaigns. We note that the production system is impossible to simulate accurately due to the many nonlinear interactions, most importantly the external influence of campaign managers, who modify constraints by hand in order to meet performance targets for individual campaigns.

As this method improves on the existing method when evaluated using the log-likelihood proxy, it is worth considering further evaluation. Further evaluation of the model without risk could be achieved by running it 'live' in parallel with the existing system, record the advertisement auction rankings for each system and analyse whether the differences are likely to positively or negatively affect the campaigns generating the highest revenue. This was not possible, as the existing ranking information for each auction is not exposed by system, and modifying this was considered too risky a change. Instead, this method was directly implemented in production on a small number of slots in order to evaluate the impact on revenue. Due to circumstances outside of our control, the volume of traffic to these slots over the test period was not sufficient to draw statistically significant

conclusions on the revenue impact.

We note that it would be interesting to re-implement the model described by Agarwal et al. [3] in the context of this dataset, but this was not considered signifiant enough to include. In the following section, we consider the variation in ad interaction probability with time of day, expected to be a significant factor influencing interaction probability.

3.5 Extensions

The results above show that a basic main effects model makes a significant improvement over the existing exchange method when using a 'fair' or comparable formulation and input data. With this demonstrated, we describe extensions to the model that would improve the practical performance significantly. Including these extensions from the start would have made direct comparison with the existing method less meaningful.

3.5.1 Interaction probability variation with time of day

As the model parameters are updated daily, and averaged over 24-hour periods, the interaction probability predicted by this model does not vary with the time of day. However, actual interaction probability, and the real-time bidding market varies significantly with time of day (see section 7.2). In order to account for this cyclical behaviour, the time of day should be directly incorporated in the model, which could be accomplished in a number of ways. One basic method would be to include an additional bias parameter in the model representing a time 'bin' within each 24-hour period. This parameter would be optimised in the same way as the other bias terms. We do not implement this extension, as recalculating the training and test data using a different time window involves building a distributed computing cluster (subsection 5.3.1). Instead, we choose to spend the time available on the methods described in the following chapters.

3.5.2 Exploration and exploitation trade-offs

We note that a situation may occur where an ad-slot pair has a high 'true' or 'potential' interaction probability, but a low estimated probability. This may result in the ad-slot pair never winning an auction and being denied the number of impression events required to 'discover' the true interaction probability. The system is then stuck in a local optimum. The problem of whether to deliberately allocate traffic to ad-slot combinations that do not have the highest estimated eCPM to gain information leading to a better outcome can be called an explore-exploit tradeoff.

As a way of performing this exploration, we wrote code to simulate an interaction probability for each impression event from the beta distribution held for each ad-slot pair, where the expected value of the beta distribution is the point estimate, and the variance represents the uncertainty. The resulting variation would result in 'less favourable' ads being displayed in some circumstances, exploring the possibility that these ads actually have a higher interaction rates than estimated. However, Google researchers provide an in-depth investigation into this issue [63]. The authors find that in a similar situation, incorporating exploration based on variance into the auction has no practical benefit and that a greedy strategy of ranking ads by the eCPM performs as well as any other possible strategy. This method was therefore not considered further.

Summary

In this chapter we mitigate many of the drawbacks of the existing method for estimating the interaction probability for an ad-slot pair discussed in section 2.3. We show how a hierarchical gamma-Poisson model provides better performance for representative ad-slot pairs, and discuss exponentially weighting past events to better capture temporal variation. As calculating the gamma-Poisson model for each ad-slot pair has high computational cost, we then propose a new model and show that this provides significantly better estimates of interaction probability than the existing method, as measured by the binomial log-likelihood. We also test this model in production in the ad exchange for a set of test slots. The proposed method is a combination of a main effects model and a beta-binomial distribution for each ad-slot pair. We discuss incremental improvements that would improve the model performance in production, the difficulties involved with evaluating any model offline in an unpredictable market environment, and the implications of the exploration-exploitation trade-off in this context.

Chapter 4

Incorporating ad-slot interaction factors

In this chapter, we investigate adding a term modeling the effect of the interaction between an ad and a slot to the main effects model described in chapter 3, a technique from the collaborative filtering field. The interaction term for an $\{ad, slot\}$ pair is given by the dot product of 'latent factor' vectors corresponding to the ad and slot, which are determined using a form of non-negative matrix factorization. Matrix factorization models use the observed ad-slot performance data to make predictions about the performance of unobserved ad-slot pairings. We first describe the motivation behind non-negative matrix factorization and the basic algorithm. We then describe two models incorporating both main effects and an interaction term, one optimised with respect to the root mean square error and the other optimised with respect to the log-likelihood. We describe the custom stochastic gradient descent implemenations in C used to solve the optimization problems and demonstrate that the interaction terms marginally improve on the main effects model alone.

4.1 Related work

Non-negative matrix factorization, which in basic terms is a factorization of a matrix V into two non-negative matrices W and H such that $V = WH^T$, describes a family of methods with many applications. Non-negative matrix factorization optimised using root-mean-square error is related to the singular value decomposition and is equivalent to a relaxed form of k-means clustering, where W contains cluster centroids and H contains cluster membership indicators. Some types of non-negative matrix factorization are instances of the probabilistic model called multinomial principal component analysis. A factorization that minimises the Kullback-Leibler divergence between WH^T and V is equivalent to probabilistic latent semantic analysis, optimised using maximum likelihood estimation. These methods are often used for analyzing and clustering textual data. We now consider non-negative matrix factorization in the context of recommendation systems. There is a significant body of work dealing with collaborative filtering in general and non-negative matrix factorization methods for this purpose in particular. Much of this work has been inspired by the Netflix Prize and the methods developed by leading teams in this competition. Koren et al. [69] provides a comprehensive overview of a recommendation system centered around this method.

In an industrial context similar to the online advertising context considered, two areas stand out. Yahoo Labs has published a group of papers on click-through rate and response prediction in the last several years. Much of this research has centered around click-through rates for content on the Yahoo main website, sponsored search advertising, and interest modeling for users having a Yahoo account. This is similar but not equivalent to the context of an exchange or demand-side platform, where there are many more ads to consider and less information on user properties is available. Agarwal and Chen [4] describe a method for advertisement response prediction that describes a combination of generalised linear models and matrix factorization.

The annual Knowledge Discovery in Databases conference, with both industrial and academic-focused tracks, sponsors an annual data-mining competition called the KDD Cup. The 2012 KDD

Cup Track 2 competition required participants to predict ad click-through rates using training and test datsets derived from search session logs from the Tencent proprietary search engine *soso.com*. The competition criteria rewarded the ranking of the advertisements rather than the true values, so some competitors used ranking approaches rather than direct click-through rate estimation. Many industry and academic teams entered the competition and published their findings, such as Wu et al. [131].

4.2 Non-negative matrix factorization

Recall from chapter 2 that the probability of an interaction given an impression event for an ad-slot pair $\{a_k, s_j\}$ is notated as p_{jk}. The expected value of this probability for all $\{a_k, s_j\}$ pairs can be visualised as a matrix $V \in \mathbb{R}^{n \times m}$ where n is the number of ads and m is the number of slots. More generally rows in V may correspond to any entity from the advertisement hierarchy, and columns of V may correspond to any entity from the slot hierarchy described in subsection 2.2.2. The entries in the matrix are the expected interaction probabilities of an impression given the ad hierarchical entity and the slot hierarchical entity for the row and column respectively.

The distributions of interaction probabilities in each row and column of the matrix V have very high variance. Put another way, the interaction probability for an impression conditional on the advertisement varies widely depending on the slot (context) on which the ad is displayed, and the interaction probability of an impression conditional on a slot varies widely with the advertisement. Possible explanations include variation in the type of user that frequents given sites; the reason the users are visiting the site (recreation, information, social, etc); the placement of the slot on the site; and similar reasons. Given these explanations, it may be possible to predict the performance of an ad on a given slot based on the performance of a similar ad on that slot. The corresponding qualitative explanation is that advertisements for airline tickets would be expected to perform better across all travel or finance-related websites than all online gaming websites.

As no descriptive information is readily available for ad and slot hierarchical entities, which could be used to infer the interaction probability, we consider inferring the matrix V using the information contained in V itself. We therefore investigate matrix factorization based collaborative filtering methods which are applicable to this use case [69]. Matrix factorization models use the observed ad-slot performance data to make predictions about the performance of unobserved ad-slot pairings. Matrix factorization has been successfully used for predictive modelling in e-commerce, with an example being the item recommendation system of Ebay which deals with similar sparsity, scale, and temporal challenges to online advertising [123].

The preference of each user for each ad can be visualised as a matrix $V \in \mathbb{R}^{n \times m}$, with the n users corresponding to the rows and the m ads corresponding to the columns. Each entry in the matrix is a real number encoding the user-ad preference, derived from the user's observed interaction, or equally, non-interaction with a given ad.

Observed values are available for some user-ad combinations in this matrix. The observed entries are a set $V_{i,j}, i, j \in \Omega$, where Ω is a subset of the complete set of $n \times m$ entries. The probability of a user interacting with an ad they have not seen is an unobserved entry in this matrix. The observed entries are a tiny fraction of the unobserved entries as most users have only seen a small subset of ads.

If the matrix V has full rank, then no model can infer missing entries. However, it is a reasonable

assumption that users can be accurately described by a number of behavioural and preferential factors much less than the total number of users. Similarly advertisements may be described by a comparatively small number of factors such as brand, type of product or service, cost, and similar. It is likely that these latent (unobserved) factors can be combined to closely estimate the user-ad preference. If these assumptions hold, then a parametric, generative model for the latent factors trained on the observed data may be able to infer the missing entries in the matrix. This is equivalent to stating that the probabilities in the full matrix are generated by a process with degrees of freedom much smaller than $n \times m$.

Let V be the user-ad matrix. Then one way of formalizing the intuition above is the statement that $\hat{V}$ can be approximated as the rank-r product of two $n \times r$ matrices W and H^T

$$V \approx \hat{V} = f(WH^T)$$
$$\text{potentially subject to } W_{ia} > 0;\ H_{bj} > 0; \forall\ i, a, b, j. \tag{4.1}$$

The columns of V and H represent the user and ad latent factors respectively. Consider the case where f is the identity function. Each element of V can be obtained by the vector dot product of the corresponding rows of W and H

$$\hat{V}_{ij} = \mathbf{w}_i^T \mathbf{h}_j$$

If the matrix V is fully observed, it is well known that the most accurate rank-r approximation with loss function given either by the spectral or by the Frobenius norm is the truncated singular value decomposition

$$M = \sum_{k \in [r]} \sigma_k u_k v_k^T$$

where $\sigma_1, ..., \sigma_r > 0$ are the singular values, and the singular vectors $u_1, ..., u_r \in \mathbb{R}^n$ and $v_1, ..., v_r \in \mathbb{R}^n$ are orthonormal vectors. For the approximation under consideration W and H are analogous to the left and right singular vectors with the singular values included into either W and/or H.

We note that Billsus and Pazzani [23] initially proposed using the singular value decomposition in a collaborative filtering context. There are many variations on the basic factorization principle presented here, across many domains. Basic variations include the choice of link function f, the definition of the loss function implied by $\approx$, and requirements placed on W and H. Criteria are such as sparsity in W or H or both are sometimes desired [61]. A common requirement is that the factors are non-negative, allowing only additive combinations. In this case, solving the specific problem $WH = V$ exactly for W and H where rank(V) is exactly k, with $k \geq 1$, is NP-hard [1],and in general non-negative factorization algorithms are non-convex with many local optima. Note that W and H are not unique as for any invertible matrix R we have $\hat{V} = WH = WR^{-1}RH$.

In the user-ad context considered in this project and collaborative filtering in general, the full matrix V is not known and the missing entries cannot be treated as zero, making the standard SVD algorithm and the many alogrithms for non-negative matrix factorization which assume V is known unsuitable. We desire an algorithm that optimises the chosen loss function over only the observed entries in V to obtain approximate candidates for W and H. Two common loss functions are the Euclidean distance (L_2 norm), which assumes additive Gaussian noise, and the generalised

Kullback-Leibler divergence (Lee and Seung, 1999) which assumes a Poisson model where the noise variance scales linearly with the model. Using the Eucludian norm, W and H are found from

$$\min_{W,H} ||V - WH||^2 + \lambda_1||W||^2 + \lambda_2||H||^2 \tag{4.2}$$

$$\text{subject to } W_{ia} \geq 0, H_{bj} \geq 0, \forall i, a, b, j \tag{4.3}$$

where the terms involving λ_1 and λ_2 are regularization parameters penalizing complexity in the solution (equivalent to a zero-mean normally distributed prior on the factors) which empirically mitigates overfitting. As these loss functions are non-convex, one class of methods involves alternately fixing W and H and solving the remaining convex optimization problem in H or W at each step. The alternating least squares algorithm is an example of a method from this class, and is more suitable for densely filled (but not complete) matrices. Another class of methods involves cycling through the rows and columns of W and H and modifying individual rows to reduce the approximation error, and is a form of stochastic gradient descent.

Non-negative matrix factorization can be seen to 'embed' the user and ad points into a new vector space, where the elements of the vectors describing each point are formed by combining the corresponding elements of the user or ad singular vectors. The coordinates of a point in this new space often have a real-world interpretation, with the value of each dimension relating to a concept such as the strength of the user preference for expensive or cheap items. If the columns of W and H are almost orthogonal it is possible to order the significance of the latent features. It is also possible to find similar users or ads by selecting other nearby points in the embedded space.

4.3 Latent factor model with the sub-optimal RMSE as the loss function

In the online advertising context, there are orders of magnitude differences in the mean conversion rate for each row and column in the input matrix, which will dominate the latent factor vectors if the algorithm above is applied directly. The factorization algorithm can be improved by the addition of bias terms similar to those discussed in the previous chapter, leading to the 'standard' matrix factorization model for collaborative filtering [68], where the matrix V is approximated as

$$\Pr(Y_{jk} = 1|s_j, a_k) = V_{ij} \coloneqq \mu + b_k + b_j + \mathbf{w}_i^T \mathbf{h}_j \tag{4.4}$$

where μ is the global offset and b_k and b_j are constant offsets or bias terms for the ad and slot (or advertiser and publisher). We note the reduction in the number of bias terms compared to the previous chapter. This is done as the model is initially trained on advertiser-publisher data, before being trained on campaign-website and ad-slot data hierarchically, when the additional bias terms are added back in. It is trivial to adjust the equations and code to include the bias terms described in chapter 3. The bias terms allow the latent factors to express the interactions desired rather than trying to account for variations in ad or slot main effects. The parameters of this model can be found by minimizing the RMSE loss function

$$\underset{b_k, b_j, \mathbf{w_i}, \mathbf{h_j}}{\operatorname{argmin}} \sum_{i,j} \left(V_{ij} - \mu - b_k - b_j - \mathbf{w}_i^T \mathbf{h}_j\right)^2 + \lambda_1 b_k^2 + \lambda_2 b_j^2 + \lambda_3 ||\mathbf{w}_i||^2 + \lambda_4 ||\mathbf{h}_j||^2 \tag{4.5}$$

Where the λ_i factors are individual regularization constants. Using individual regularization constants results in empirically better results [69]. As the output values from this model are not constrained to the range $[0, 1]$, the output is truncated to lie between 0 and the 95^{th} percentile value of the observed advertiser-publisher ad interaction rates.

Recall from section 2.2 that the existing exchange/demand side platform method for interaction probability estimation does not account for the impression context (the slot) other than at the most specific level in the hierarchical MLE process. Almost any model that includes the context contribution to the interaction probability is likely to improve the log-likelihood significantly over the existing model. The bias-only model is therefore used as the baseline model with which to evaluate the factorization models involving ad-slot interactions rather than the existing method of section 2.2.

4.3.1 Temporal variation and low-confidence observed rates

Temporal variation is accounted for by using the exponentially-discounted maximum likelihood estimate $\hat{p}_{jk}$ described in subsection 3.2.2 to form the training data for the model. This is a different and simpler approach to the standard practice of using time-varying bias and interaction terms in the main effects/latent factor models. We avoid introducing this complexity while we are still evaluating the utility of the interaction terms in the first place.

The SVD-based model is trained with the matrix $\hat{V}$ where the entries in the matrix are the maximum likelihood estimates of the interaction rate for that row and column. As the average value of $\hat{p}_{jk}$ is on the order of 10^{-4} the maximum likelihood estimate has a wide confidence interval where few events have been observed for the row and column. In order to deal with this, a number of events greater than a minimum threshold of 5000 are considered for both training and testing the model. For an ad-slot pair with an interaction probability equal to the overall mean, the maximum likelihood estimate of the interaction probability given a sample of 5000 impressions has an 87% chance of being within 80% of the true value (by a simple application of the binomial distribution), a threshold which is arbitrarily chosen.

4.3.2 Optimization

Stochastic gradient descent with early stopping was used to solve the convex optimization problem, learning the offset terms and latent feature vectors. The squared error for an individual term in the training matrix is

$$e_{ij}^2 = \left(V_{ij} - \mu - b_k - b_j - \sum_{k=1}^{K} w_{ik} h_{kj} \right)^2 + \lambda_1 b_k^2 + \lambda_2 b_j^2 + \lambda_3 \sum_{k=1}^{K} w_{ik}^2 + \lambda_4 \sum_{k=1}^{K} h_{kj}^2$$

where K is the number of latent factors (the number of columns in matrices W and H) and $\mathbf{w}$ and $\mathbf{h}$ are the latent feature vectors. For each training example, we loop through the parameters in (4.5), updating each parameter in the opposite direction to the gradient, using the following update

equations

$$b_k \leftarrow b_k + \alpha_2 \frac{\partial}{\partial b_k} e_{ij}^2 = b_k + \alpha_2 \left(e_{ij} - \lambda_1 b_k\right)$$
$$b_j \leftarrow b_j + \alpha_3 \frac{\partial}{\partial b_j} e_{ij}^2 = b_j + \alpha_3 \left(e_{ij} - \lambda_2 b_j\right)$$
$$\forall k \in \{0, 1, ..., r\} \;\; w_{ik} \leftarrow w_{ik} + \alpha_4 \frac{\partial}{\partial w_{ik}} e_{ij}^2 = w_{ik} + \alpha_4 \left(e_{ij} h_{kj} - \lambda_3 w_{ik}\right)$$
$$\forall k \in \{0, 1, ..., r\} \;\; h_{kj} \leftarrow h_{kj} + \alpha_5 \frac{\partial}{\partial h_{kj}} e_{ij}^2 = h_{kj} + \alpha_5 \left(e_{ij} w_{ik} - \lambda_4 h_{kj}\right)$$

where the α_i terms are the learning rates for each parameter, r is the number of components in each latent feature vector, and the λ_i terms are the regularization terms for each parameter. We loop through the training examples multiple times, until the error on a validation set (held out from the training set) starts increasing due to overfitting.

This solver was prototyped in Matlab and Python but was ported to C due to lack of speed reducing the amount of hyperparameter optimization possible. Hyperparameters include the initial latent factor values, which are initialised with random vectors drawn from a Gaussian distribution, the learning rates, regularization constants, and the order with which the training examples are processed. A basic grid search was conducted for the optimum hyperparameter values.

4.4 Latent factor model with log-likelihood as the loss function

The RMSE has serious drawbacks as a loss function in this context, as discussed in section 2.4 and section 4.6. We therefore use the logistic loss function as an optimization criteria for the interaction model. We note that this has previously been proposed in [83]. In this method the probability $\hat{p}_{jk} = \hat{V}_{ij}$ is given by

$$\hat{p}_{jk} = \hat{V}_{ij} \coloneqq \sigma(\mu + b_k + b_j + \mathbf{w}_i^T \mathbf{h}_j) \tag{4.6}$$

where $\sigma(\cdot)$ is the logistic log function. Learning the parameters is done by minimising the log-likelihood, yielding the following optimization problem

$$\underset{b_j, b_k, \mathbf{w_i}, \mathbf{h_j}}{argmin} \sum_{j,k} -T_{jk} \log \hat{p}_{jk} \; - \; (T_{jk} - S_{jk}) \log(1 - \hat{p}_{jk}) + \lambda_1 b_k^2 + \lambda_2 b_j^2 + \lambda_3 ||\mathbf{w}_i||^2 + \lambda_4 ||\mathbf{h}_j||^2 \tag{4.7}$$

In this equation, T_{jk} is the number of impressions for a given advertiser-publisher pair combination, and S_{jk} is the number of ad interactions for the combination.

This model has the properties of weighting the loss function by the relative confidence in the observed value of each entry [1], as the maximum likelihood estimation of the true value of the assumed generative model becomes more accurate with increasing impressions. Also, the model's use of the logistic function bounds the output within [0,1]. The same method for mitigating the effect of temporal variation in p_{jk} as before is also used with the log-linear factorization method.

4.4.1 Optimization

A stochastic gradient descent implementation similar to the implementation described in subsection 4.3.2 was implemented in C++ using the following update rules

$$a \coloneqq \mu + b_k + b_j + \mathbf{w}_i^T \mathbf{h}_j$$

$$b_k \leftarrow b_k + \alpha_2 \left(\frac{e^a T_{jk}}{1 - e^a} - S_{jk} \right) + 2\lambda_1 b_k$$

$$b_j \leftarrow b_j + \alpha_3 \left(\frac{e^a T_{jk}}{1 - e^a} - S_{jk} \right) + 2\lambda_2 b_j$$

$$w_{ik} \leftarrow w_{ik} + \alpha_4 h_{jk} \left(\frac{e^a T_{jk}}{1 - e^a} - S_{jk} \right) + 2\lambda_3 w_{ik}$$

$$h_{kj} \leftarrow h_{kj} + \alpha_5 h_{jk} \left(\frac{e^a T_{jk}}{1 - e^a} - S_{jk} \right) + 2\lambda_4 h_{kj}$$

We also experimented with using a FORTRAN implementation of the L-BFGS algorithm [85] to find a local minimum of this optimization problem. This limited-memory BFGS algorithm is a quasi-Newton method using Shanno-Phua scaling to compute the step direction and a bracketing line-search for a point satisfying the strong Wolfe conditions to compute the step length. However, this produced inferior results to the custom stochastic gradient descent implementation.

4.5 Hierarchical factorization models

If the matrix V is constructed using ads and slots, the number of observed entries is small compared to the unobserved entries due to the data sparsity - most ads have only been displayed on a small subset of slots. Beyond a certain sparsity, it is not possible to recover low-rank struture within V even if it exists.

There are well researched limits on the fraction of entries required for exact recovery of a low-rank matrix [29], a similar situation to the matrix factorization algorithm considered here. Clearly at least $(n_1 + n_2 - r)r$ measurements are required to exactly recover a matrix V where $\text{rank}(V) = r$, where n_1 and n_2 are the number of ads and slots respectively. Exact recovery is possible using a number of measurements within a constant of this limit via convex programming. However, the observed ad-slot interaction matrix data does not meet the minimum recovery requirement above.

Therefore, a hierarchy of factorization models are constructed. The training data is first aggregated to the advertiser-publisher level, and a factorization model is trained on conversion rate data at this level of the hierarchy. A second factorization model is then constructed at the campaign-website level, and trained on the **residual** between the prediction of the first model and the training data at the campaign-website level. A final factorization model is then constructed at the ad-slot level, and trained on the **residual** between the prediction of the second model and the training data at ad-slot level.

4.6 Results and discussion

We train the models on exchange data from the period of June 2013. The model is tested on its ability to predict aggregate interaction rates for ad-slot pairs for a 'future' dataset from the first

week of July and also from the entire month of July. The results from the first week are reported, as the model can readily be trained on a single machine in several hours, making it feasible to re-train every day. This is a true test in the sense that the prediction of the model is being validated against unseen future data, in contrast to reporting the ability of the model to fit the training data. There are $8,330$ advertiser-publisher training pairs in the dataset out of $582,000$ possible pairs. There are $6,300$ pairs in the test set, including 855 pairs (14%) unobserved in the training set.

The log-likelihood is the primary measure of model quality. The model RMSE are reported as it is the metric the SVD-based factorization model is trained on. We note that the distribution of true p_{jk} values over the matrix V is approximately a gamma distribution with the majority of values being on the order of 10^{-3} to 10^{-4} with a long tail of much higher values. The RMSE is dominated by these high click-through rate (CTR) outliers just as the gradient of a linear regression is heavily influenced by high leverage outliers. In addition, the RMSE does not consider the difference in the number of events per ad-slot pair. For instance, a pair with 2 impressions and 1 click has the very high CTR of 0.5 but is unlikely to be as relevant as a pair with 10^6 impressions and 100 clicks. The log-likelihood captures this variation. However, both the log likelihood and the RMSE place high penalties on occasional large mispredictions and small penalties on the much larger volume of small mispredictions. Given the distribution of interaction probabilities in the ad auction it may be more beneficial to overall revenue to place more significance on penalizing the large number of smaller mispredictions near the mean value than the occasional large misprediction. The performance of each model is given in Table 4.1, Table 4.2, and Table 4.3.

Section	Model	RMSE	LL
	bias terms only	0.38151	-1690163
section 4.3	log-linear	0.37274	-1608127
section 4.3,subsection 4.3.1	log-linear time-discounted	0.37263	**-1591549**
section 4.4	SVD-based	0.26449	-1678458
section 4.4,subsection 4.3.1	SVD-based time-discounted	**0.25196**	-1618914

Table 4.1: Factorization model performance at the advertiser-publisher level

Model	RMSE	LL
bias terms only	1.06162	-1690163
log-linear time-discounted	1.01887	-1547982
SVD-based time-discounted	0.70282	-1574431

Table 4.2: Factorization model performance at the campaign-website level

Model	RMSE	LL	AIC
existing exchange method	1.21346	-1,890,565	3,689,293
bias terms only	1.36578	-1,518,400	3,071,583
log-linear time-discounted	1.20424	-1,491,265	3,589,188
SVD-based time-discounted	1.10742	-1,509,876	3,589,188

Table 4.3: Factorization model performance at the ad-slot level

The SVD-based model decreases RMSE significantly, which is expected as this is directly optimised by the algorithm over the training set. The log-linear factorization method produces the best

results in terms of the log-likelihood, significantly improving on the baseline model, and making a minor improvement over the bias term model. The bias terms are responsible for the majority of the performance of each model with the interaction terms in each factorization method providing marginal additional benefit. The time-discounting of older events in order to capture variation in interaction probability over time provides a consistent benefit to the model. Following this investigation, the interaction terms, or latent vectors, are not considered to be useful as the decrease in log-likelihood is not large enough to offset the additional parameters introduced into the model.

The gradient descent algorithm produced better results than L-BFGS optimization, producing the best result for each method after multiple runs varying the parameter initializations. This is attributed to the ability to individually vary the learning rates and regularization for each parameter with the gradient descent method, as well as the ease of early stopping in order to avoid overfitting.

Other than the event aggregation into the advertiser-publisher training information, which is also performed for other purposes, this model is not computationally demanding in this context, being able to be trained on a single machine in hours. It is noted that in the online setting of the exchange, new information is continually arriving. The matrix factorization model can be updated efficiently by minimising the objective function considering only constant and latent factors that affect updated entries in the matrix V_{ij}, and these updates can be performed in parallel.

Summary

In this chapter, we investigate modeling the effect of interactions between hierarchical interaction between an ad and a slot to the main effects model described in chapter 3, a technique from the collaborative filtering field. The interaction term for an $\{ad, slot\}$ pair is given by the dot product of 'latent factor' vectors corresponding to the ad and slot, which are determined using a form of non-negative matrix factorization. We first describe the motivation behind non-negative matrix factorization and the basic algorithm. We then describe two models incorporating both main effects and an interaction term, one optimised with respect to the root mean square error and the other optimised with respect to the log-likelihood. We describe the custom stochastic gradient descent implemenations in C used to solve the optimization problems and demonstrate that the interaction terms marginally improve on the main effects model alone. However, the interaction models described were not found to be suitable tools for interaction probability estimation.

Chapter 5

Incorporating user-specific features with generalised linear models

In this chapter we model the user-ad interaction probability $Pr(Y_{hjkt} = 1|u_h, s_j, a_k, t)$ as a function of advertisement, user, context, and time features using a regularised generalised linear model. We first develop a production-ready data pipeline based on the Hadoop software for processing impression handler log files described in subsection 2.5.2 and generating the various processes required for the model input features. We then use a logit-linked generalised linear model of the interaction probability as a function of these features, constructing a separate model for each campaign. After experimenting with this model, we develop a scalable implementation that is capable of training thousands of campaign models over hundred of millions of impressions daily, and evaluate the predictive performance using an industry dataset containing approximately 9×10^9 events. All information used in this chapter is anonymised and cannot be used to personally identify any user when the dataset is taken in isolation.

5.1 Related work

Generalised linear models and gradient descent methods are widely used in computational advertising by demand-side platforms and exchanges due to their scalability and performance. Researchers working with industry have contributed significantly to the academic literature due to the commercial advantages from improvements in accuracy and training efficiency. As this thesis seeks to apply recent research to an industrial problem, the following paragraphs focus on industry contributions to the academic literature, rather than one or more specific methods for determining the parameters of a generalised linear model (Agarwal et al. [2] and Ross et al. [112] are two good papers on this topic). The predictive model and infrastructure developed in this chapter is similar to those described in the papers referenced in the paragraphs below, although significantly less complex. All of the industrial systems described have access to many more features than are available in this project, as data collection has not been a focus of the exchange. We also note that many of these papers were published in the past two years.

A regularised generalised linear model is used for ad interaction prediction in the 2013 paper [81], describing work done at Google. A second paper from engineers at Google containing practical and general advice for machine learning in large, complex production systems is [116]. In this paper the authors suggest incorporating as many features as possible into a generalised linear model and relying on strong L_1 regularization to enforce parameter sparsity. The authors also present a map-reduce based implementation for training models with stochastic gradient descent that inspired the method developed in subsection 5.4.2 below.

Collective is a New York based company that operates a large demand-side platform in the U.S. market. In January 2014, Collective released a paper describing some aspects of their advertisement targeting and value estimation system [64]. To model user-campaign interaction probability, Collec-

tive uses per-campaign logistic regression models with L_1 and L_2 regularization. After computing a per-campaign model, Collective groups users by thresholding their model score, and targets each campaign to the user groups most likely to interact with the campaign. Collective uses hundreds of thousands of user features, obtained from data vendors as well as cookie-based tracking. We note that Collective, as a demand-side platform, does not bid on all users available through exchanges and supply-side platforms, choosing only users that meet model data requirements and a threshold probability of interacting with a given campaign. We note that this is a different situation to the exchange setting where it is recalled that a fixed catalogue of ads must be optimally assigned to all impression inventory.

Distillery (previously Media6Degrees) is an online targeted advertising company based in the US. Distillery captures an extensive and informative range of data including physical location, device, and mobile data. Distillery uses this data to build brand-specific user interaction models, and then displays targeted advertisements to users across multiple internet-connected platforms including television and mobile devices [103]. Distillery's process includes segmenting users, and then using the user segments as features in an unspecific. Distillery user segments for each campaign typically account for about 1% of all users, and the literature does not provide any details about how this user segmentation is accomplished. The output of the generalised linear model is then used in further model layers to determine the bid price for an ad impression. Distillery only bids on around 3% of available impression inventory. Distillery describes using unique URLs visited by a user as a set of user features, the same idea we developed in this project.

Microsoft has both display and search advertising operations, and has undertaken a significant amount of published and unpublished research into estimating ad interaction probability [110]. A notable paper is that by Graepel et al. [52], which describes an extremely scalable probit (not logistic) regression model for interaction probability prediction, notable by its use at scale in the Bing search engine. In this Bayesian model, parameters are described by distributions which are updated in real time as new training examples are observed. Another notable paper by researchers affiliated with Microsoft research describes the 'orthant-wise limited-memory quasi-Newton' algorithm, a modified L-BFGS method that allows scalable training of an L_1-Regularised generalised linear model [10].

Finally, Criteo is another major competitor in the U.S. and global online advertising industry. A paper by Chapelle et al. [33] provides some details on a CTR prediction method used by Criteo. This method is also based on L_1 and L_2 regularised generalised linear models.

5.2 Model evaluation criteria

We repeat section 2.4 here to allow for the situation where this chapter is read independently.

The ideal evaluation criteria for any interaction probability estimation method developed in this project is that it generates more revenue than the existing method, all other factors being equal. The only definitive way of testing for this criteria is to run each system in parallel and direct ad requests to each system at random. However, in order to evaluate candidate methods offline, a proxy for this evaluation method is required that can be used in isolation with historical data. Developing an accurate and precise proxy is difficult, as small changes to the predicted interaction probabilities at time t affect future auction outcomes. We use the the binomial log-likelihood calculated over all

impression events as a proxy. The log-likelihood is defined by

$$l(\phi) = \sum_{i=1}^{N} \left(y_i \log(f(\phi)) + (1 - y_i) \log(1 - f(\phi)) \right) \tag{5.1}$$

where N is the number of test events, $y_i \in \{0, 1\}$ is the ocurrence or non-occurrence of an interaction for an impression event, and $f(\phi)$ is the prediction function for the interaction probability.

We evaluate proposed and existing models by training each model on a historical dataset from the exchange and then comparing the log-likelihood for the interaction probabilities generated by each model for held-out test data from after the training period. We note that using the log-likelihood for evaluating the performance of an interaction probability prediction system in the context of an exchange or demand-side platform suffers from a number of shortcomings. Most significantly, the binomial log-likelihood penalises large errors to a greater extent than small errors, on a continuous scale. However, the outcome of an ad auction is a discrete event, and a small error which changes the auction ranking has a greater impact than a large error which does not change the ranking and auction outcome. However, developing a more application-specific method is more time-consuming than simply testing the new method on a small fraction of live traffic.

We also note that in the situation where a model is used to predict interaction probabilities for an ad-slot pair the root-mean-square error (RMSE) between predicted and actual interaction probability is a very poor error metric. It assumes that the errors are all from a normal distribution, which is a terrible assumption for count data; Poisson is a reasonable assumption. The RMSE also does not account for the number of impressions for each ad-slot pair, unlike the log-likelihood. It is dominated by low-confidence outliers in a similar fashion to ordinary linear regression. The RMSE also heavily penalises large errors while minimally penalizing small errors (although the log likelihood does this also). Given the distribution of interaction probabilities in the ad auction it may be more beneficial to overall revenue to place a higher penalty on the large number of smaller mispredictions near the mean value than the occasional large misprediction.

5.3 Feature engineering and data processing

Recall that given an incoming ad request from a publisher or real-time bid request, we require an estimate of the interaction probability for each advertisement in the exchange or demand-side platform catalogue in order to select the 'best' advertisement as defined in section 2.1. Rather than using aggregate historical interaction rates as in chapter 3, we now model the probability of an ad interaction taking place following an impression event as a function of information related to the event (see Table 5.1). We use logistic regression, a regularised generalised linear model with a binomially distributed response variable and a logit link function. This model can be trained using historical impression events which are served directly through the exchange or via the real-time bidding market. We choose to construct a separate model for each campaign. For a given campaign, this model constitutes a mapping from a real-valued vector representing all the information describing an event to a probability of an interaction occurring. In order to learn this mapping, we first require a method for transforming information about the event to a real-valued feature vector denoted by $\mathbf{x_i}$, a process known as feature engineering.

Information, or features, relating to an event can be usefully grouped into information relating to the advertisement, user, direct context, and external context of an impression event. These

groups are shown in Table 5.1. We note that all of the information used comes from the event logs described in subsection 2.5.2. Information not included in these log files, including basic information such as the semantic content of advertisements and websites, is not available in the context of this project; obtaining this information is highly desirable and the subject of ongoing work.

Type	Features
User	Browsing history, device
Direct context	Publisher, website and slot
General context	Time of day

Table 5.1: Categories of features used

Information relating to the user includes the user's available browsing history, defined in this project as the number of times each user has requested each webpage tracked by the exchange, including aggregate features such as the total number of websites visited and the total number of website requests. The motivation for including these features is that the browsing patterns contained in this information will differentiate users with respect to their ad interaction behaviour [78]. The user's computing device, determined by the browser user-agent string, is also included in the model. The users' IP address, which can be used to determine physical location, is not included, although it would be simple to include in a future version. Information relating to the general context of an impression event includes the time of day. The time of day of an ad impression event is highly correlated with the interaction probability, and is therefore included in the feature vector. Information relating to the direct context of an impression event included in the model includes the slot, website, and publisher.

We note that features can be both categorical or continuous. Categorical features with N possible values are represented by a 1-in-N encoding over multiple dimensions, one dimension for each level of the categorical variable, with one of these dimensions equal to 1 and the remainder equal to 0, also known as a 'one-hot' binary encoding. We also note that for a proportion of impression events, the user involved has not been previously identified by the exchange. For these impressions, the feature vector cannot be generated, as no previous impressions exist. We therefore consider only impression events where the user has been previously seen in 3 or more events for both training and testing the model, as we consider that these users also have an acceptable probability of being seen again in a relevant time period.

Prior to using the event logs to generate the features described above, the logs must be processed to filter out search engine crawlers, automated agents, and fraudulent clicks. After these events are filtered out, only events reasonably believed to correspond to people interacting with online content remain. We used a set of heuristic filters based on exploratory data analysis to identify and remove these events. While we are reasonably confident our methods eliminate the majority of such traffic and result in useful training data, we note that this is an active research domain in its own right.

5.3.1 Hadoop and mapreduce

We now consider generating a processed dataset consisting of the features described above from the textual event logs, with one event per line. As the exchange processes billions of events daily, these files, even in compressed form, represent a large volume of data: around 270 billion lines, or 3 Terabytes of uncompressed text, were processed to generate the results in this thesis. Processing the

events to generate features, creating model input matrices, and training models over this volume of data would be too time-consuming on a single workstation, and we therefore used multiple virtual machines running the Hadoop and MPI distributed computing frameworks.

Datacenter-based virtual machines were used in place of physical servers. The virtual machine (VM) instances were rented on a per-hour basis from the Amazon Web Services Elastic Compute Cloud (EC2). EC2 is a core product of Amazon Web Services (AWS), which offers other more integrated products such as the distributed database Redshift. It is possible to start, stop, and terminate these instances when desired, paying by the hour for active machines - hence the term 'elastic'. VM specifications and costs change on a frequent basis and are available from the AWS website. A range of servers are available, ranging from single-core instances with 500MB of memory to 16-core instances with 244GB of memory (at the time of writing). We experimented with different cloud instance types to find the most cost-effective instance size for the feature engineering, settling on more instances of the m1.xlarge VM type rather than fewer instances of a larger VM type. This is expected to be because the workloads do not require large Java Virtual Machine heap sizes and greater IO capacity is available with greater numbers of smaller instances. The virtual machines were hosted in the AWS Ireland datacenter, the closest datacenter to the exchange location. We note that renting VMs by the hour is more economic than purchasing machines for experimental work or projects with varying computational requirements. For example, for this project we ran a 100 machine cluster with 400 cores, 1.5 TB of RAM, and 168 TB of disk space with a hardware cost of around $500,000 for 24 hours for a cost of under $\$5,000$ (one of many configurations). Further benefits of cloud instances include easy access to different operating systems and compilers, and ephemeral environments with little system administration overhead. While working on this project, we developed a library of scripts to automate starting, stopping and configuring different size clusters for different workloads.

Storing and processing data on a cluster of machines requires a software framework to organize computational tasks, manage data locality, and manage redundancy to compensate for hardware failure. Machine failure is nearly certain, as a 24-hour computation depending on the output of 100 machines has a 40% probability of failure if each machine has a 0.5% probability of failure in the same period. The software framework must therefore silently handle the loss of multiple machines without losing data or affecting the progress of a program. Two requirements are a distributed filesystem and a system for managing parallel computation between machines that does not depend on high-speed networking. One such framework is named Hadoop. In 2006, Google published solutions to these problems in [31] and [44], and these were replicated in the open-source Hadoop project in the following years by engineers at Yahoo and elsewhere.

The Hadoop distributed filesystem (HDFS) allows all machines in a cluster to access files stored on any other machine, as well as providing configurable data replication for redundancy. The distributed computation abstraction provided by the first-generation Hadoop framework is called mapreduce [44]. The mapreduce framework requires the user to write programs which follow the steps illustrated in Figure 5.1.

1. Split the input data into $(\text{key}, \text{value})$ pairs with an arbitrary function - typically by line, although method of splitting is not required
2. Process the $(\text{key}, \text{value})$ pairs with an arbitrary function called the 'map' function
3. Sort the $(\text{key}, \text{value})$ pairs and write to HDFS

4. Fetch the (key, value) pairs from their location in the cluster
5. Merge the (key, value) pairs into one sorted file
6. Read these files sequentially and process them with an arbitrary function called the 'reduce' function
7. Write the output of the reduce functions to HDFS

Crucially, writing software using this abstraction allows the map and reduce functions to be performed by multiple processes in parallel, as there is no communication between functions during their execution. Therefore, the number of map and reduce processes can be increased arbitrarily, and the volume of data that can be processed scales almost linearly with the available computing resources. Multiple iterations of this process allow common data processing operations such as aggregations, joins, filters, and transformations to be performed in parallel using thousands of servers if required.

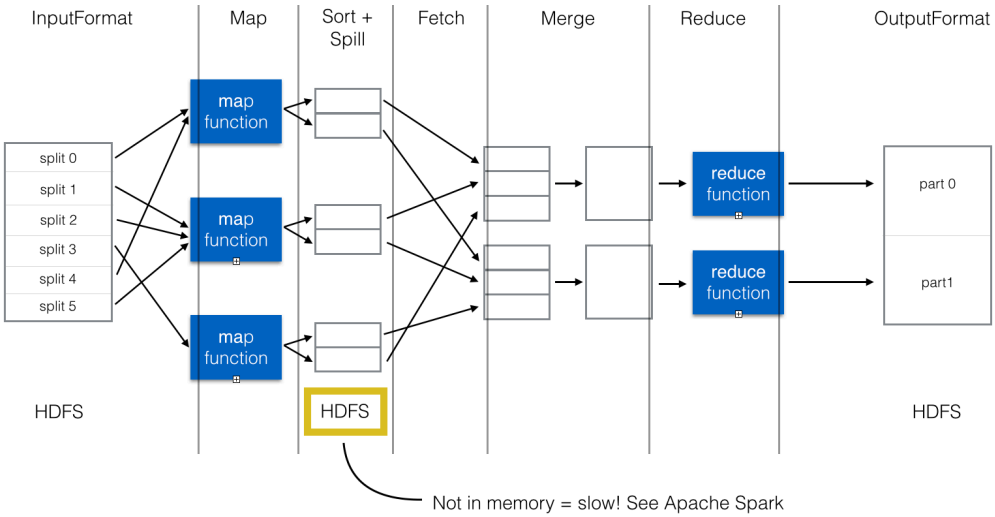


Figure 5.1: Conceptual illustration of the Hadoop mapreduce process

One example of the many uses of the mapreduce abstraction in this project is calculating the time that each unique user in the dataset was first observed and last observed. For each event, the map function outputs the user identifier as the key and the timestamp as the value. The reduce function processes all events for a given user and selects the smallest and largest timestamps observed. The time required for this calculation was found to exhibit almost linear scaling with both the number of events and the cluster size, in line with expectations. A second example is the calculation of the number of impression and interaction events for the ad and slot entities and ad-slot pairs in chapter 3 and chapter 4, which is a trivial application of the mapreduce paradigm. A further example of the use of the mapreduce framework is generating the portions of the event feature vectors corresponding to the user-website interactions. The map phase of this computation partitions the lines in the log files into $\{key, value\}$ pairs where the user identifier is the key and the website identifier is the value. The reduce phase of the computation aggregates the number of website identifiers for each user.

Mapreduce is a powerful abstraction, particularly for simple data transformations built from aggregation, filtering, and similar steps. However, iterative methods for function minimisation and algorithms that require graph traversal are not a natural fit with the mapreduce system. A further

limitation of Hadoop and mapreduce is that the time required to load data from the servers' physical disks, move data around over the network, and serialize intermediate results to and from disk, is often more limiting than processing capacity. A framework that is superceding mapreduce for many workflows due to its ability to cache intermediate results in memory is the Apache Spark framework, which we do not investigate here due to time limitations.

5.4 The generalised linear model

Following the data pre-processing, information describing each ad request is represented by a sparse vector $\mathbf{x_i} \in \mathbb{R}^{n+1}$, where n is the number of features or dimensions and the extra dimension is a constant value representing the model intercept. The outcome of each ad request (an interaction or no interaction) is denoted by y. The pair $(\mathbf{x}_i, y)$ is an event. We model the occurrence of an ad interaction occurring as a result of an impression event using a generalised linear model and the logistic link function. From this point on we denote the occurrence or non-occurrence of an interaction event for an impression event by $y \in \{-1, +1\}$ for notational convenience. The sampling distribution of a logistic regression model is then

$$\Pr(y|\mathbf{x}, \mathbf{w}) \coloneqq \sigma(\mathbf{w}^T\mathbf{x}) \tag{5.2}$$

where σ is the logistic function and $\mathbf{w} \in \mathbb{R}^{n+1}$ is the model weight vector. We note that the probit link function is an alternative to the logistic link function in this context. The parameter vector is found by minimising the negative log likelihood

$$l(\mathbf{w}) \coloneqq \sum_{i=1}^{N} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda_1||\mathbf{w}||_1 + \lambda_2||\mathbf{w}||_2 \tag{5.3}$$

where λ_1 and λ_2 are the model regularization constants, known as L_1 and L_2 regularization constants. The linear L_1 penalty term corresponds to a Laplacian prior on the model coefficients, written as

$$\Pr(\mathbf{w}) = (\lambda_1/2)^n \exp(-\lambda_1||\mathbf{w}||_1) \tag{5.4}$$

In practice, the L_1 term can reduce coefficients which have little effect on the log-likelihood of the model to exactly zero, depending on the optimization method and implementation used. L_1 regularization allows fitting a model where the number of degrees of freedom approaches or exceeds the number of training examples, as the sample size required grows logarithmically in the number of irrelevant features [91]. However, the L_1 regularised objective function is not differentiable everywhere, requiring special treatment by gradient-based solvers. The quadratic L_2 penalty, corresponding to a Gaussian prior on the coefficients, does not result in zero coefficients, and is differentiable everywhere. L_2 regularization requires a sample size that grows linearly in the number of irrelevant features. The regularization parameters penalize the size of the coefficients, reducing the tendency of the model to overfit. Overfitting is when the function learned by the model produces a low error on the training dataset but does not generalise to new data from the same distribution, due to the model fitting 'noise'.

The vast majority of impression events do not result in any user interaction (the average interaction rate is around 10^{-4}), including most impression events where the user is *a priori* known to have

a high affinity for the advertisement in question. Therefore, we expect that the positive interaction events contain more information than negative interaction events, and that increasing the relative importance of the positive training events may improve the classifier performance. Weighting the positive training examples more heavily than the negative training examples can be achieved by duplicating each positive training example h times, where h is equal to the desired (integer) weighting factor.

Alternatively, this can be achieved by subsampled negative training examples, a process which typically has negligible impact on the model error when working with extremely large datasets. A side effect of this process is reducing the computational complexity due to the order of magnitude decrease in size in the dataset, which reduces the server costs. We therefore downsample the training data by using a heuristically chosen ratio r smaller than 1. This technique is common in the literature and employed in production by many other firms, including the leading demand-side platform Distillery [103]. We note that this requires adjusting the model by adding $\log(r)$ to the intercept to account for the altered probability distribution after subsampling [66], [96], [32].

A common step in statistical modelling is feature selection. However, standard techniques for estimating the significance of each feature are difficult to apply in this context, with thousands of models being simultaneously trained on the same extremely large dataset. One practical approach would be to build a base model for each of the thousands of campaigns with a set of basic features, then train a new model for each campaign using a set of additional features, and then evaluate the candidate feature set using the mutual information criterion with a held-out test set over all campaigns. However, feature selection using this process has not been performed due to time constraints, and we include all features in the modelling process by default.

5.4.1 Minimising the logistic regression loss function

The demand-side platform considered in this project can process up to billions of impression events daily. Due to the volume of training records, a fast and memory-efficient method is required to minimize the loss function (5.3) for each campaign. Stochastic gradient or coordinate descent and L-BFGS optimization are two obvious candidates, and we note that these methods can be modified to account for the non-differentiability of the L_1 regularization term for some values of the parameter vector using subgradient methods [10], [73]. We select stochastic gradient descent due to its conceptual simplicity and the desirable property of only requiring the parameter vector and one training record to be held in memory at a time.

Stochastic gradient descent considers one randomly selected training record at a time and updates the parameter vector $\mathbf{w}$ using the gradient of the loss function with respect to the parameter vector $\mathbf{w}$ at that training example. In general terms, the parameter vector is updated according to

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \vec{\nabla}_w g(\mathbf{x}_i, \mathbf{w_i}, y_i) \tag{5.5}$$

for each training example, where $\vec{\nabla}_w$ is the gradient operator with respect to $\mathbf{w}$, γ_t is the learning rate or gain, and g is the logistic loss function (5.3). Gradient descent, unlike stochastic gradient descent, updates the parameter vector by considering the gradient of the objective function with respect to all training examples. Coordinate descent, on the other hand, reduces the objective function by conducting line searches along each coodinate direction in iterative fashion. The convergence properties of the stochastic gradient descent have been studied extensively in stochastic

approximation literature, and the method converges with high probability under mild conditions [26]. We note that the optimization error (which is reduced by gradient descent) is distinct from the approximation error (which is how closely the generalised linear model can approximate the optimal prediction function) and the estimation error (which is the impact of minimizing the loss over the training sample, the empirical loss as oppposed to the population or expected loss) [126]. We also note that for extremely large datasets, stochastic gradient descent can be distributed across multiple machines by updating local parameter vector on each machine and synchronizing with a master parameter vector at defined intervals [136]. We do not require this approach as we train one model per campaign, and the training data for each campaign is small enough to process sequentially.

Due to the commercial relevance of scalable linear solvers, a number of companies have developed highly optimized libraries, and some have been made publicly available. The stochastic gradient descent implementation in the vowpal wabbit library [65] was used in this project to evaluate the model and features described above. On current hardware, vowpal wabbit is typically bound by disk input/output speed rather than CPU speed, and the software has the capability to run in a distributed environment. Vowpal wabbit also employs many sophisticated methods in pursuit of low memory demand, computational efficiency, and speed of convergence. These include feature hashing, automatically evaluating the current solution against a held-out test set from within the training data to guard against overfitting, and an optimization method which does not require normalizing of the feature vectors [45] to avoid the regularization terms penalizing some coefficients more than others. We note that the FORTRAN stochastic gradient descent implementation wrapped by the **glmnet** function in R would be an alternative for this purpose.

Hyperparameters for the vowpal wabbit implementation of stochastic gradient descent include the learning rate, the number of passes over the training data set, the order in which the training examples are provided to the learner, and the L_1 and L_2 regularization constants. A limited grid search was conducted to optimize the hyperparameters during experimentation.

5.4.2 Training thousands of models over billions of events

We now describe implementing this method in a way that is sufficiently scalable and robust for production use. Recall from section 5.3 that a separate generalised linear model is constructed for each campaign. Each campaign model requires updating after a given volume of new impression events have occurred, which is typically every 1-7 days. Updating (or recalculating) a model requires processing the event logs into the form required for model input, training the model, and storing the output for each of thousands of campaigns.

Due to the number of models required, the requirement for hardware-fault tolerance and ease of administration, and the fact that data preprocessing is performed in Hadoop, it makes sense to use the mapreduce framework to orchestrate the model training process. In order to train the models, we developed a mapreduce job where the map phase partitions the data by `campaignid` and sends all records for each campaign to a single reducer, which performs the online stochastic gradient descent. We modified an open-source Java implementation of stochastic gradient descent [11] to run within the reduce functions, as the *vowpal wabbit* library is written in C++ which is difficult to integrate into the Java Virtual Machine runtime environment of the Hadoop framework. We fit models with both regularised and non-regularised feature vectors $\mathbf{x}$, as this implementation is sensitive to normalization.

5.5 Combining the hierarchical and generalised linear models

While the model described above is designed to produce an estimate of the interaction probability which does not require further modification, it is significantly more expensive to compute than the main effects model developed in chapter 3, and the data pipeline supporting the model has not been proven to be stable in production. It may not be used for all campaigns initially. Also, the main effects model is conceptually similar to the existing exchange method and easier to reason about than the generalised linear model. For these reasons, it is desirable to develop a method for transitioning smoothly to the generalised linear model that is not a sharp switch-over.

We developed the following principled way to combine the output of the generalised linear model above with the main effects model. We desire a function g to combine the two models given by

$$\Pr(Y_{hjk}|\mathbf{x}, s_j, a_k) = g(f(\mathbf{x}), p_{jk}) \tag{5.6}$$

where $f(\mathbf{x})$ is the output of the logistic regression model and p_{jk} is the output of the main effects model. In order to construct g, we choose to break impressions into segments denoted z using the distribution of the generalised linear model output $f(\mathbf{x})$ over the training data. We then define g as

$$g(f(\mathbf{x}), p_{jk}) = \phi_z p_{jk} \tag{5.7}$$

where ϕ_z is a correction factor for the segment z. Letting E_{z,s_j,a_k} be the expected number of interactions (successes) given T_{z,s_j,a_k} impressions (trials) for ad a_k, slot s_j, and segment z as determined by the main effects model, and we propose to determine S_{z,s_j,a_k}, the expected number of interactions (successes) for the segment as

$$S_{z,s_j,a_k}|E_{z,s_j,a_k}, \phi_z \sim \text{Poisson}(E_{z,s_j,a_k}\phi_z) \tag{5.8}$$

where the value of ϕ_z is determined by minimizing the log-likelihood

$$l(\phi_z) = \sum_{z,s_j,a_k} (-E_{z,s_j,a_k}\phi_z + \log(\phi_z)S_{z,s_j,a_k}) + \text{a constant} \tag{5.9}$$

This is computationally achievable as the number of segments z is chosen to be relatively small (e.g. 20). Also, if the generalised linear model is mis-configured and provides no information, then $\phi_z = 1$ and the combined model is equivalent to the main effects model. Increasing or decreasing the influence of the generalised linear model is possible by tuning the value of ϕ_z manually.

To implement the system we propose above, we run another map-reduce job following the job which trains the generalised linear models, in order to generate a model prediction for each line of the log file (log line), followed by a mapreduce job to assign a user segment to each line. Finally, a mapreduce job is run to aggregate the impression log lines over the (a_k, s_j, z) groups to generate training input for the correction factor model above. Determining the value of the correction factors ϕ_z can be performed on a single machine using common optimization libraries in reasonable time.

5.6 Making predictions in real time

Once the models are trained, we require a method for integrating the interaction probability predictions made by the stand-alone modeling system into the complex, distributed architecture of the exchange/demand-side platform system. Computing estimated interaction probabilities in real time has demanding latency and throughput requirements, on the order of thousands of requests per second with a maximum response time of approximately 40 milliseconds. Minimal modification of the existing exchange infrastructure is desirable, in order to facilitate gradual switch-over and simple rollback.

We propose implementing the method described in this chapter by creating an independent system similar to a real-time bidding platform. When an ad request is received, the exchange can request the interaction probability correction factor ϕ_z for each ad from this independent system given the adrequest $= s_j, u_h$, and use these correction factors to modify the auction. The proposed architecture of the system is shown in Figure 5.2, taken from a project planning document.

The first component of this system is a cluster of servers to handle the modification factor request, denoted 'IBID' in Figure 5.2. These servers would run an API server written in Node.js or similar, or a modification of full-scale open source real-time bidding client RTBKit, written in C++. The second component is a distributed, in-memory key-value store of the correction factors. The servers hold the model coefficients for each campaign in memory. As modification factor request arrives, the servers query a low-latency data store for the information stored against the entities in the request and form the feature vector $\mathbf{x}$. This vector and the model coefficient vector are then used to generate the model output, and the result is returned to the main adservers conducting the auction. Simultaneously, the user data store is updated with the new user information derived from the ad request. This process is abstracted behind an API which is queried by the ad impression handler (ad server), gaining implementation flexibility at the expense of slightly higher latency.

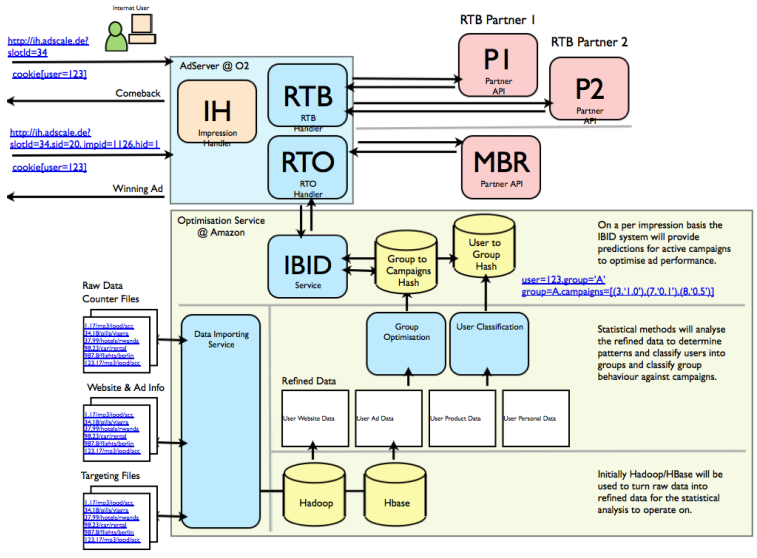


Figure 5.2: Proposed architecture of the real-time interaction probability estimation system

We experimented with some components of this system, including the low-latency in-memory key-value store. We set up a two-node installation of the Couchbase database, using `c3.xlarge` AWS servers, and achieved a throughput of over 20,000 user documents per second (latency is not directly comparable as the networking enviornment of any production system would be different

to this test environment). This was mainly for interest as Couchbase meets these requirements in many existing installations.

5.7 Results and discussion

5.7.1 Improvement in log-likelihood for individual campaigns

The campaign model method above was evaluated using a dataset of all events for the 96 largest campaigns by click volume occurring in the exchange for the month of August 2013. While these campaigns are not a representative sample from all campaigns managed by the exchange at any point in time, they are deliberately selected as they represent a non-negligible fraction of interaction-based revenue. Any system considered for implementation is required to produce adequate results on these campaigns, and even if the system does not give increased performance on other campaigns, it would still be worth implementing. The event data for each campaign is split into a training set and a testing set in an 80/20 ratio.

The performance of the generalised linear model is evaluated using the improvement in the log-likelihood for the test dataset with respect to a baseline consisting of the main effects model developd in chapter 3. This baseline is chosen as it is the criteria for considering the model for commercial use - it must deliver enough marginal revenue over the best alternative method to justify the significant increase in computational expense. Figure 5.3 shows a histogram of the percentage improvement in log-likelihood for the 96 test campaigns compared to the main effects model. An improvement in the log-likelihood value over the main effects model is seen for all but one campaign.

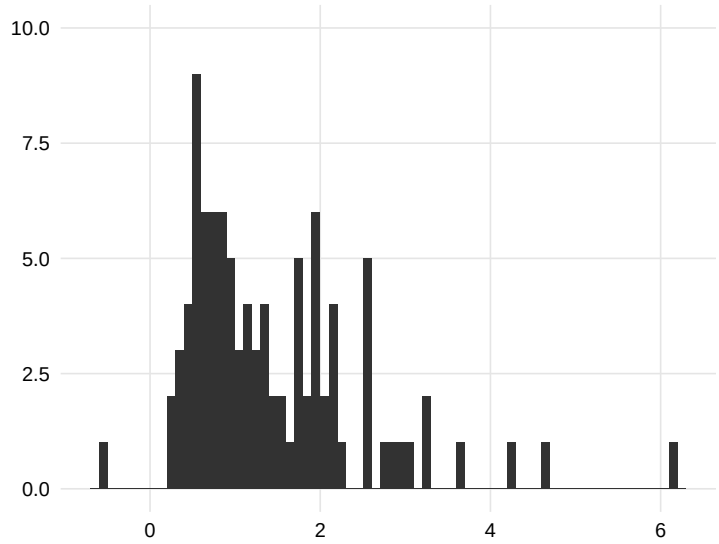


Figure 5.3: Histogram of the percentage increase in log likelihood for the 96 test campaigns (approximately 50 million impressions). The horizontal axis is the percentage increase and the vertical axis is the number of campaigns.

We further illustrate the performance of the logistic regression model by separating the events in the test set into four segments *for each campaign* and calculating the mean interaction rate for each segment. We then normalize the mean segment interaction probability with respect to the mean campaign interaction probability for all events in the test set. These normalized click-through rates are displayed in Figure 5.4. To illustrate this figure further: if events are assigned to a segment

at random, the mean interaction probability for the segment would be expected to equal the mean interaction probability for the campaign as a whole. Alternatively, if the bins correspond to events with different underlying properties, then the mean interaction probability for each segment is expected to differ from the mean interaction probability for the campaign as a whole. The figure below displays the distribution of the normalized click-through rates for all 96 test campaigns for each of the 4 bins. The segment with the lowest click-through rate has, on average, a maximum likelihood (mean) interaction rate half that of the campaign mean. The segment with the highest click-through rate has, on average, a mean interaction rate around 1.3 times that of the campaign mean interaction rate.

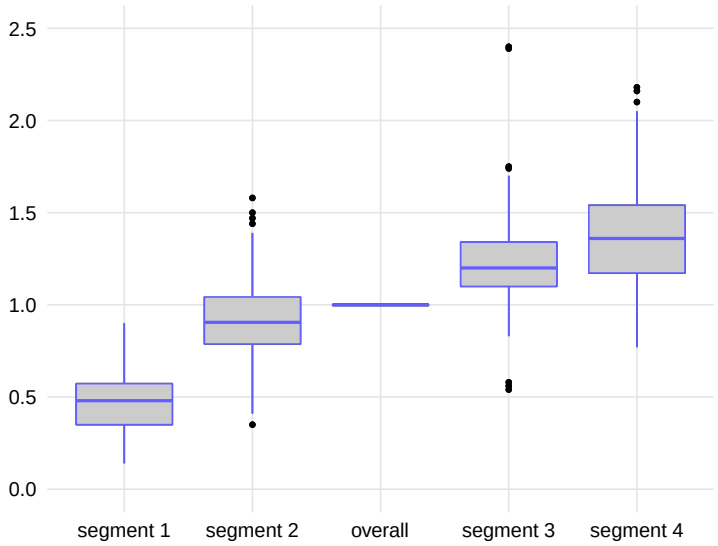


Figure 5.4: Distribution of segment interaction probabilities for all of the 96 test campaigns, where the interaction rate of each segment is expressed as a ratio of the campaign average. The vertical axis is the segment interaction rate expressed as a ratio of the campaign average; therefore, if Segment 1 for a given campaign has a value of 0.5, it means that the segment has an interaction rate half that of the campaign average.

We did not find a significant difference in the model performance for normalized or non-normalized feature vectors. We note that Google researchers also found no improvement in the performance of a large scale online logistic regression model from normalizing feature vectors as $\mathbf{x}/\text{norm}(\mathbf{x})$ for a variety of norms [81], a similar setting to that considered here.

5.7.2 Ability to differentiate user-campaign affinity

It is possible that the campaign-specific models above are merely identifying an overall higher probability of campaign interaction for a user, rather than identifying specific preferences for individual campaigns. This is valuable given the presence of both CPA/CPC and CPM campaigns in an auction, as modifying the interaction probability only affects the eCPM value for CPA and CPC campaigns. However, if all campaigns were CPA campaigns, inferring an overall higher probability with no campaign-specific element would affect all ads in the auction equally, and offer no benefit at all. This is a significant concern, and is not dealt with in the previous section.

Therefore, we selected 10 random pairs of campaigns from the 96 test campaigns and re-trained the model above on all selected campaigns, using training and test datasets consisting only of events for which the users in the events had been exposed to both campaigns. We then compared the events

falling into the bins described above for each campaign in a pair. On average, 15% of events in the top 50% for one campaign were in the bottom 50% for the other campaign. This indicates that this modeling system has the ability to differentiate between campaigns to a degree likely to be significant in a CPA-only environment.

As a further test, we constructed a training and test dataset consisting of only positive user-ad interaction events. For each campaign in the 96 test campaigns, we trained a logistic regression model attempting to differentiate users who had interacted with that particular campaign from users who had interacted with other campaigns. This model was able to achieve approximately 15% higher log-likelihood than random assignment, providing further empirical validation of this framework.

5.7.3 Improvements

Incorporating more informative features is a significant area to focus on for model improvement. More and more informative data is typically superior to a more complex model. Useful information includes user information such as physical location history, e-commerce product view and purchase data, and advertisement and website semantic properites. The model is also expected to perform significantly better with more sophisticated data pre-processing than performed in this project, as the feature distributions used are noisy and highly skewed. Higher performance would be achieved by conducting feature selection and remvoing unnecessary features, a process we do not perform due to lack of time to set up the processing jobs required. The model would also benefit from hyperparameter optimization, including the subsampling ratio and regularization constants. Independent variable removal may also improve model performance, and could be accomplished by removing blocks of variables and determining the resulting change in performance.

5.8 Investigating third-party cookie alternatives

We note that user-based features depend on the ability of the exchange to identify individual users across multiple ad requests or real-time bid requests. On receiving an ad request for a given user, the exchange/demand-side platform servers attempt to place a unique identifier into the user's browser. If the identifier is already present, the user has previously been seen and the exchange server reads the identifier back. This unique identifier, typically a third-party cookie, has a limited lifespan, as it is frequently removed when users clear their browser history. Individual users can therefore be tracked for variable periods of time before the unique identifier is erased from the user's browser.

The identifier written by the exchange servers into a user's browser is the best means available to relate an ad request to am individual person making the request. This identifier is typically a 3rd-party tracking cookie, although more recent methods such as browser fingerprinting or HTML5 local storage are becoming increasingly common in order to circumvent users removing or blocking cookies. The identifier, rather than the user, is the entity tracked by the exchange. If a person has more than one computer or browser (e.g. tablets and phones) or multiple people share a computer (such as in a family home) then a one-to-one mapping will not exist between a person and a browser identifier.

Cookies have limited lifespans, and are becoming less useful over time as more users clear their browser history and block 3rd-party tracking cookies. Some browsers prevent 3rd-party cookie placement entirely (such as Mobile Safari) and others are becoming less open to 3rd-party tracking

cookies, such as the recent proposal by the developer of the popular Firefox browser to block these cookies by default.

In order to mitigate this problem, developers at the exchange implemented an alternative unique identifier called an etag. The difference in lifespan distribution between cookies and etags was investigated using the infrastructure developed during this project for user targeting. After placing both identifiers together for approximately a month, the lifetimes of individual identifiers using the new method were found to not be significantly better than cookies. Generating these lifetime curves involved processing over a terabyte of log data on a 40-machine cluster.

Total Cookie Persistant Users $>$ 1 Day Duration	42108145
Total Etag Persistant Users $>$ 1 Day Duration	47471223
Comscore March Figures	44500000

Table 5.2: Number of unique users

A survival curve for standard tracking cookies and etags is shown in Figure 5.5. Etags offer slightly improved performance but do not offer substantial increase in tracking effectiveness.

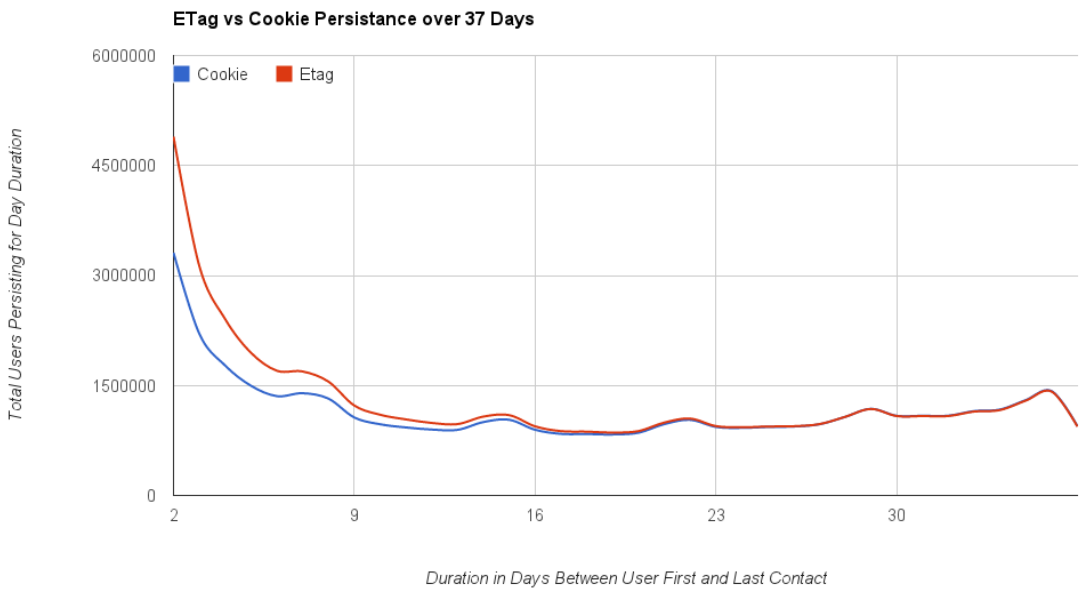


Figure 5.5: Etag vs standard cookie survival analysis for the Chrome browser. The upper line corresponds to the etags and the lower line corresponds to the standard cookies.

Summary

In this chapter, we develop, validate, and then implement a system which uses regularised generalised linear models to estimate ad interaction probability on a per-event level. We first develop a distributed data pipeline for processing raw impression logs into a dataset ready for model training using the Hadoop framework. This process involves filtering automated agents and crawlers from the logs, selecting persistent users and removing single-session users, creating all required model features, and normalizing the resulting matrices. We then develop scalable, distributed system for

training a generalised linear model for each campaign, also using the Hadoop framework. These systems can process hundreds of millions of impressions on a daily cycle. We then evaluate the predictive performance offline using an industry dataset containing approximately 9×10^9 events, and find a small but commercially significant improvement in the predictive performance.

Chapter 6

Manifold regularization, dimensionailty reduction, and graph segmentation

The methods and software developed in chapter 3 and chapter 5 generate significantly better results than the existing ad-interaction probability estimation method used by the exchange and demand-side platform, and are suitable for production use with minimal modification. As part of this project, we desired to study high-performance distributed computing, spectral graph theory, stochastic block models, clustering algorithms, the solution of large eigenproblems, and dimension reduction techniques. We therefore use the data set and problem from chapter 5 as context for conducting experiments in these areas. We emphasize that the work in this chapter was focused on gaining experience with computational methods and large datasets, and was not expected to be useful in the industrial context. We therefore give less attention to the performance in estimating user-ad interaction probability than in previous chapters.

We first investigate manifold regularization of a linear least-squares model. We then use the Laplacian eigenmap method to embedd vectors representing each unique user into a new space, and use the user vectors from this space as additional features in the generalised linear model considered in chapter 5. We then investigate clustering unique users using spectral clustering and modularity maximisation methods, and investigate the difference in aggregate user-campaign interaction probability between clusters. We also investigate clustering the k-nearest neighbour user graph and the bipartite user-website graph by fitting degree-corrected stochastic block models.

6.1 Manifold regularization

We now consider calculating a function that maps *user* vectors to a probability of the user interacting with a campaign at some point in the future, a different problem statement than that considered in chapter 5. We select a subset of the user features from the feature engineering process, and create a data matrix X where each row corresponds to a user, and each column corresponds to a website from the websites observed by the exchange, described as the user-website feature space. If a user has visited a website, the number of visits is entered into the appropriate row and column in X. We normalise X by column for all of the numerical experiments below.

Given a set of users and their outcome for a given campaign — either an interaction or no interaction — $(\mathbf{v}_i, y_i), i = 1...L$, an unknown function mapping the feature space $\mathbf{V}$ to a probability of an interaction can be written as [20]

$$f^* = \underset{f \epsilon \mathcal{H}_K}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} V(\mathbf{v}_i, y_i, f) + \gamma_A ||f||_K^2 \tag{6.1}$$

where V is a loss function such as the squared loss, K is a Mercer kernel $K : \mathbf{V} \times \mathbf{V} \to \mathbb{R}$, $\mathcal{H}_K$ is a Reproducing Kernel Hilbert Space (RKHS) of functions $f : X \to \mathbb{R}$ with a norm $||.||_K$, and γ_A is a regularization constant.

The number of users who have been exposed to a given campaign, noted as L, is small compared to the overall number of unique users, noted as N. It may be possible to estimate an improved function for a given campaign by incorporating knowledge of the distribution of all event vectors, in addition to the event vectors for which a campaign interaction has been observed. In order to investigate this assumption we assume that events with a higher probability of interaction with a given campaign are similar to one another and different to events with a lower probability of interaction, where similarity is defined by a given distance metric.

For a given campaign, user-campaign interactions $(\mathbf{x_i}, y_i)$, $\mathbf{x_i} \in X$ are drawn from the probability distribution P on $X \times \mathbb{R}$, where X is the space spanned by the user vectors. We assume that the conditional probability distribution $\mathcal{P}(y_i|\mathbf{x}_i)$ is a smooth function of the marginal distribution $\mathcal{P}_X$, and that if two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are "near" under some distance metric in the user-website space X, then the conditional distributions of campaign interaction probability $\mathcal{P}(y_i|\mathbf{x}_i)$ and $\mathcal{P}(y_j|\mathbf{x}_j)$ are similar. Users for which no campaign interaction has been observed are users drawn from the marginal distribution $\mathcal{P}_X$ [20].

Given these assumptions, an additional regularization term $\gamma_I||f||_I^2$ can be added to (6.1) penalizing lack of smoothness in f with respect to $\mathcal{P}_X$, [20] leading to

$$f^* = \underset{f \epsilon \mathcal{H}_K}{\operatorname{argmin}} \frac{1}{L} \sum_{i=1}^{L} V(x_i, y_i, f) + \gamma_A||f||_K^2 + \gamma_I||f||_I^2 \tag{6.2}$$

We now further assume that the support of $\mathcal{P}_X$ is approximately a compact submanifold. This assumption is intuitively appealing, as it is unlikely that the browsing patterns of large numbers of users will explore the entire user-website space uniformly. If the additional regularization term above is chosen to penalize sharp changes in gradient along geodesics of the manifold corresponding to $\mathcal{P}_X$ [20], then (6.2) becomes

$$f^* = \underset{f \epsilon \mathcal{H}_K}{\operatorname{argmin}} \frac{1}{L} \sum_{i=1}^{L} V(x_i, y_i, f) + \gamma_A||f||_K^2 + \gamma_I \int_{x \in \mathcal{M}} ||\nabla_{\mathcal{M}} f||^2 d\mathcal{P}_X(x) \tag{6.3}$$

where $\nabla_{\mathcal{M}}$ is the gradient of f along the manifold $\mathcal{M}$. The manifold is unknown, but can be approximated by the symmetric, positive semidefinite graph Laplacian constructed from the labelled and unlabelled data points with exponential weights [55]. The positive, semidefinite graph Laplacian is defined as

$$L = D - S \tag{6.4}$$

where D is the diagonal matrix given by

$$D_{ii} = \sum_{j=1}^{N} S_{ij} \tag{6.5}$$

and S is the pairwise similarity (or adjacency) Gram matrix. Letting $\hat{f} = [f(x_i), ..., f(x_N)]^T$ the optimization problem becomes

$$f^* = \underset{f \epsilon \mathcal{H}_K}{\operatorname{argmin}} \frac{1}{L} \sum_{i=1}^{L} V(x_i, y_i, f) + \gamma_A||f||_K^2 + \frac{\gamma_I}{(u+l)^2} \hat{f}^T L \hat{f} \tag{6.6}$$

As for the kernel support vector machine, the Representer theorem [**?**] can be used to show that the function f^* exists in $\mathcal{H}_K$ and has the following form

$$f^*(x) = \sum_{i=1}^{N} \alpha_i^* K(x_i, x) = \alpha^* K \tag{6.7}$$

where $K(x_i, x)$ is a Mercer kernel and K is the $N \times N$ Gram matrix corresponding to the data. The additional regularization term can therefore be described as 'warping' the kernel function according to the geometry of $\mathcal{P}_X$. Expressing f as an expansion of kernel functions in $\mathcal{H}_K$ (the dual form) results in the ability to learn a nonlinear function in the original space. This is the basis of the well-known Support Vector Machine along with kernel ridge regression. Belkin et al. [20] derive the equations required to solve a Laplacian regularized version of the kernel support vector machine. Solving this problem is difficult for large n as the naive solution requires the inversion of the dense Gram matrix [119], [41], whose computational cost is $O(n^3)$, although Chen et al. [35] discuss ways of working around this limitation and Tsang and Kwok [125] develop a sparse manifold regularizer and apply their method to a dataset of a million examples. An appropriate kernel function and bandwidth must also be selected. If f is restricted to be a linear function, then (6.6) becomes

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{L} (y_i - \mathbf{w}\mathbf{x}_i)^2 + \lambda ||\mathbf{w}||_2 + \gamma_I' \mathbf{w}^T X^T L X \mathbf{w} \tag{6.8}$$

and it is possible to find f^* by solving this convex optimization problem, (where a constant has been included in the vector x similar to the previous chapter) [120]. Setting the gradient to zero leading to the following linear system

$$(X_L^T X_L + \gamma_A L I + \gamma_I L X^T L X)\mathbf{w} = X_L^T \mathbf{y} \tag{6.9}$$

where the $L \times L$ matrix X_L is the matrix of user vectors for which a campaign interaction has been observed, and is a submatrix of the $n \times n$ matrix X. $\mathbf{y}$ is the vector of corresponding interactions (1 for a positive interaction and 0 for a negative interaction).

6.1.1 Linear manifold regularization experiments

As an experiment, we tested this method with a subset of the 96 campaigns considered in the previous chapter. In this context, X is very sparse (for every event, most features are zero) and L is also sparse as only the k nearest neighbours are used in the construction of the Laplacian matrix associated with the data (section 6.2). The conjugate gradient algorithm, a Krylov method for finding an approximate solution to a linear system $Ax = b$ is therefore appropriate for solving this system. We used the implemention of this algorithm in the PETSc software [13].

This least squares method does not output a bounded result that can be interpreted as a probability, unlike the generalised linear model discussed previously. In order to make predictions with the output of this model, we segment the users by model score as discussed in subsection 5.7.1 and section 5.5. We evaluate the model for the labelled users in the training set, divide the users into a number of segments using the score, and calculate the maximum likelihood interaction probability for each segment. The model can then be evaluated for each event in the test set. The test users are assigned a appropriate segment and the maximum likelihood probability for that segment is used

as the model prediction.

This method failed to produce better results than the generalised linear model described in the previous chapter for any of the subsets of campaigns tested. This was not unexpected, as the model uses the squared loss rather than maximum likelihood loss as in logistic regression, which is a less appropriate loss function for binary count data.

6.2 Spectral embedding of the user vectors

Rather than attempting to find (6.1), we now consider calculating a nonlinear projection of the user vectors into a lower dimensional space using the Laplacian eigenmap method. We select a subset of the user features from the feature space discussed in chapter 5, and create a matrix X where each row corresponds to a user, and each column corresponds to a website from the websites observed by the exchange. If a user has visited a website, the number of visits is entered into the appropriate row and column in X. We propose to project into a lower-dimensional space that preserves locality on the manifold $\mathcal{M}$ discussed above. If the embedding preserves spatial relationships in the lower-dimensional projection, then this new feature space may provide additional information that can be utilized in a generalised linear model alongside the user-website feature space to estimate the probability of a user interacting with a campaign.

A drawback of the embedding method described in this section is that it does not provide a means of embedding out-of-sample user vectors, which eliminates the practical utility of this model for ad interaction probability estimation. In this context, new users continually arrive, accumulate new features over time, and disappear from the dataset. An embedding calculated at time t does not provide an embedding for points appearing from t onwards, and embedding these points requires re-calculating a new embedding. As the intent was to investigate the properties of the embedding using a static dataset, and not apply it in practice, this limitation was ignored. However, we do investigate embedding out-of-sample users at the same location as their nearest neighbor in the original space.

Belkin and Niyogi [18] show that the function $f : \mathcal{M} \to \mathbb{R}^n$ from the smooth, compact manifold $\mathcal{M}$ to $\mathbb{R}$ that best preserves locality on average is given by minimizing the squared gradient on the manifold

$$\begin{aligned} \underset{f \epsilon \mathcal{H}_K}{\text{argmin}} \quad & \int_{x \in \mathcal{M}} ||\nabla_{\mathcal{M}} f||^2 d\mathcal{P}_X(x) \\ \text{subject to} \quad & ||f||_{L^2(\mathcal{M})} = 1 \end{aligned} \tag{6.10}$$

which is the last term in Equation 6.8. If the manifold is again approximated by the graph Laplacian, the 'best' locality-preserving map of the user vectors to the real line is found by the optimization problem

$$\begin{aligned} \underset{\mathbf{y}}{\text{argmin}} \quad & \mathbf{y}^T L \mathbf{y} \\ \text{subject to} \quad & \mathbf{y}^T D \mathbf{y} = 1, \\ & \mathbf{y}^T D \mathbf{1} = 0. \end{aligned} \tag{6.11}$$

where $\mathbf{y}$ is the embedding of the user vectors on the real line, the condition $\mathbf{y}^T D \mathbf{y} = 1$ eliminates a scaling factor in the embedding and the constraint $\mathbf{y}^T D \mathbf{1} = 0$ eliminates the trivial solution $\mathbf{y} = \mathbf{1}$. The vector $\mathbf{y}$ which solves the optimization problem is the Fiedler eigenvector of the generalised eigenvalue problem $L\mathbf{y} = \lambda D\mathbf{y}$. Extending the mapping from the real line to $\mathbb{R}^k$ is done by assigning

each data point $\mathbf{x}_i$ to the point in $\mathbb{R}^k$ defined by the i-th entries in the k eigenvectors corresponding to the smallest eigenvalues λ, $\lambda \neq 0$ of the generalised eigenvalue problem $L\mathbf{y} = \lambda D\mathbf{y}$ [18], [19]. This technique is sometimes called the Laplacian eigenmap, and has been shown to be equivalent to kernel Principal Component Analysis, where the Gram matrix is equivalent to the affinity matrix up to a normalization constant. This technique is closely related to the family of algorithms described as spectral clustering.

We note that if the graph Laplacian embedding method is performed with the weighted dot product graph (adjacency matrix) rather than the Laplacian matrix, Sussman et al. [124] show that a k-nearest neighbours classifier based on the embedding converges to the Bayes optimal classifier. There is a large body of work on theoretical and practical aspects of spectral embedding and Laplacian eigenmaps.

We now describe the methods required to conduct this spectral embedding process for the user vectors generated in chapter 5. First we consider computing the sparse adjacency matrix of the k-nearest neighbour user graph, from which the graph Laplacian and normalized Laplacian matrices are found. We then consider solving the large eigenproblem $L\mathbf{y} = \lambda D\mathbf{y}$. We note that other methods of nonlinear dimension reduction exist, but are not investigated due to time constraints. These include locally linear embedding and stochastic neighbour embedding.

6.2.1 Constructing the k-nearest-neighbour graph

Computing the graph Laplacian matrix naively requires calculating $n \times n$ pairwise similarities between n user vectors. In the online advertising context, n is larger than 10^7, making this naive calculation expensive and almost certainly uneconomic. The dense similarity matrix alone requires over 2350 TB of storage. Therefore, a fast method for approximating the graph Laplacian is required. It is possible to approximate the Laplacian using the Nystrom method [47]; however, we choose to approximate the Laplacian by computing and storing only the top k pairwise similarities in the pairwise similarity matrix S [18], Song et al. [121]. Other methods for selecting a subset of k pairwise similarities include selecting only those similarities above a threshold, or selecting entries at random, or variations of these methods. We do not consider these methods as they typically result in a disconnected graph. The sparse pairwise k-nearest-neighbour method is not symmetric, as a k nearest neighbour relationship between $\mathbf{x}_i$ and $\mathbf{x}_j$ does not imply the converse. The matrices are made symmetric by setting every entry S_{ij} equal to S_{ji} where S_{ji} or $S_{ij} \neq 0$. The number of adjacent nodes for each node in the adjacency matrix does not have more than $2k$ entries in any row following the process performed to make the matrix symmetric. We now consider methods for calculating the similarity matrix, using the normalized user-website feature space.

6.2.1.1 Similarity functions

The Gaussian similarity function (the heat kernel) for two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is

$$s_{ij} = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{6.12}$$

where σ is the kernel bandwidth. This choice of metric results in the graph Laplacian being asymptotically convergent to the Laplace-Beltrami operator, desirable for approximating the manifold as

described above. Another potential similarity function is the cosine similarity

$$s_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{||\mathbf{x}_i||||\mathbf{x}_j||} \tag{6.13}$$

which calculates the angle between two vectors x_i and x_j. While this similarity metric completely changes the geometric interpretation of the similarity matrix S, and throws away much of the manifold-related justifications above, we consider it as it is used with success in many domains dealing with sparse, high-dimensional vectors, such as text analysis.

6.2.1.2 k-nearest neighbour implementations and experiments

We now investigate the maximum practical number of user vectors from chapter 5 for which a similarity graph can be constructed with a reasonable computational budget. As the user dataset is highly sparse, we also investigate a dense dataset as a comparison for interest. The dense dataset is a representation of all the articles in the English wikipedia as dense 500-dimensional vectors calculated using latent semantic indexing [59]. To generate the latent vectors, the raw Wikipedia text data was downloaded and the vectors computed using the open-source Gensim package [106] running on a large EC2 server, a process which took around 12 hours. The test dataset properties are given in Table 6.1.

	Wikipedia	User vectors
Number of vectors	3 651 422	up to 31 000 000
Number of dimensions	500	approx. 5000
Non-zero entries	100%	0.15% non-zero

Table 6.1: Test datasets

We used a mid 2011 Macbook Pro for prototyping and a cluster of 10 `cc2.8xlarge` EC2 instances where appropriate for experimenting with larger datasets. The cluster has a combined 605 GB of RAM and 160 Intel Xeon E5-2670 CPU cores. In order to work with numerical arrays similar in size to the Macbook's RAM we implemented custom functions in Python and C++ to load and save numerical arrays to binary and text files, as the standard functions for loading data in scientific computing libraries typically have peak memory requirements around twice the size of the array to be read. We would have liked to experiment with a GPU or Xeon Phi™ coprocessor but did not due to time constraints.

6.2.1.3 Naive k-nearest neighbour computation

We first implemented a 'naive' exhaustive pairwise k-nearest-neighbour calculation in C++. This implementation stores the input user vectors in memory in sparse format (even the fully dense Wikipedia dataset) and calculates all pairwise similarities for each input vector, maintaining the current top 10 similarities in a max-heap to faster insertion of each calculated similarity. It can use an arbitrary similarity function, as the function is passed a pair of sparse vectors. As this is a naive, exhaustive implementation, the k-nearest neighbour calculation can be done independently for each input vector, and we therefore distribute the calculation using the MPICH implementation of the Message Passing Interface standard. This implementation has a computational cost of approximately $O(n^2d + n^2\log k)$ and a storage cost of $O(nk)$ where d is the dimension of the user

vector, n is the number of input vectors, and k is the number of nearest neighbours. We note that Hadoop-based implementations of this calculation have been published [43], but we wished to experiment with MPI.

This implementation is limited by both CPU capacity and memory bandwidth (the difference in speed between cosine and Gaussian similarity functions is approximately 9%). Due to the memory controller and CPU cache design in modern systems, accessing data stored in RAM sequentially is much faster than random access Alabduljalil et al. [6]. However, the in-memory representation of the sparse format used results in non-sequential memory access. As calculating the cosine similarity is equivalent to L_2 normalizing the rows of the user vector matrix X, computing the cosine similarity matris S is equivalent to taking the matrix product $X^T X$ and selecting the top 10 values from each row. We therefore experiment with using the Intel Math Kernel Library calculate $X^T X$. This requires X to be stored in dense format, which is typically too large for RAM, and we partition X by rows into multiple submatrices, and operate on these submatrices as described in [106]. This implementation outperformed the 'naive' sparse implementation by a factor of 5 on the dense Wikipedia dataset, as expected. For the user vector dataset, the cache-optimized implementation also (slightly) out-performs the sparse implementation, despite performing an order of magnitude more operations.

To achieve faster execution for the sparse user dataset while still obtaining an exact result, we also experimented with eliminating candidate vectors for each input vector using an inverted index. This principle is the basis of information retrieval systems. This method exploits properties of the cosine similarity, and is not suitable for the Gaussian similarity function. The inverted index maps each dimension of the input vector space to the vectors which have a non-zero value for that dimension. Only overlapping non-zero dimensions then need be compared for any input pair. Bayardo et al. [15] describe extensions to this method such as the use of a threshold to further prune the number of candidate vectors, building the index incrementally, and storing the vector weights in the index. We modified a parallel implementation of this technique due to Awekar and Samatova [12], which was kindly provided by the authors. Alabduljalil et al. [7] show that under certain conditions this method is competitive in time with approximate random projection methods.

The maximum number of input vectors for which the sparse similarity matrix can be calculated within one day with these implementations and the 10-server cluster is around 2×10^6. Given the cost of this cluster this is uneconomic, and we consider ways of speeding this calculation up further.

6.2.1.4 Sub-quadratic scaling with locality-sensitive hashing and tree-based algorithms

The constant factors and quadratic scaling of the implementations described above makes finding the exact solution to the k-nearest neighbour problem for values of n larger than a few million impractical, even on the 10-machine cluster. A large body of work exists dealing with finding probabilistic or approximate sub-quadratic algorithms for this problem. Older methods based on space-partitioning trees include vantage-point trees [132], B-trees [16], and cover trees [22]. However, these methods display poor performance in high-dimensional spaces. More recently, effective methods have been developed. Some of these methods are variations on the locality sensitive hashing principle, mitigating the disadvantages of the 'vanilla' method, such as the LSH forest algorithm (ref), or a related method which constructs a forest of trees using random hyperplanes to divide the dataset at each node. Silpa-Anan and Hartley [118] construct a forest of randomized $k-d$ trees [118], one of the best performing methods at the time of writing [86]. Another excellent method

conducts a priority search on a hierarchical k-means tree [87].

The basic locality-sensitive hashing (LSH) method is a probabilistic method for finding a set of approximate nearest neighbours for an input vector with varying theoretical gaurantees. It constructs one or more hash functions $g : \mathbb{R}^d \to U$ such that for any two vectors $\mathbf{x}_i, \mathbf{x}_j$ if the cosine similarity is greater than an arbitrary value r_1, then $\Pr(g(\mathbf{x}_i) = g(\mathbf{x}_j))$ is significant, and if the cosine similarity is less than an arbitrary value r_2, $\Pr(g(\mathbf{x}_i) = g(\mathbf{x}_j))$ is insignificant. The function g can then be used to compute the approximate k-nearest neighbours of an input vector x_i by retrieving all vectors that fall into the same 'bucket' or have the same value of $g(\mathbf{x})$ as $\mathbf{x}_i$ and then computing the top k similarities from within these vectors. For the cosine similarity, a simple function that can be used to construct g is given by

$$g_i(\mathbf{x}) \coloneqq [h_1(\mathbf{x}), ..., h_m(\mathbf{x})] \tag{6.14}$$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \cdot \mathbf{r} \leq 0; \\ 0 & \text{otherwise} \end{cases} \tag{6.15}$$

where $\mathbf{r}$ is a vector of the same dimension as $\mathbf{x}$ with each component drawn from the standard normal distribution [34]. This is equivalent to selecting a number of random hyperplanes passing through the origin and assigning each user vector either a 0 or 1 depending on which side of the hyperplane they fall. This is the 'vanilla' LSH method for the cosine similarity, where each bucket is defined by a binary string.

For interest, we implemented the 'vanilla' probabilistic locality sensitive hashing algorithm using a hash function built from random hyperplane projections, and quickly ran into the well-known drawbacks of this method. In particular, the hash function is blind to the data distribution. We found that most data points occupy relatively few values of g, or buckets, with the remainder of the data points occupying buckets with less than k other occupants. No amount of tuning parameters such as the number of hash functions or random projections alleviated the strongly non-uniform distribution of input vectors across hash buckets. Various authors have improved on 'vanilla' LSH with with techniques such as probing multiple buckets (binary strings) with low Hamming distance, or constructing a prefix-tree from the hash codes and probing nearby leaves of the tree [14]. Other works improving on the basic LSH method include [97], [135], [9], and [94]. A different and very interesting probabilistic algorithm that is independent of the input vector dimension, allowing millions of dimensions to be used, is given by Zadeh and Goel [133], as well as the basics of a map-reduce based implemenetation.

Given the unsuitability of the vanilla method we used the elegant `annoy` library [21] for experimenting with the full dataset. This library creates a forest of trees across the input vectors (with a fixed leaf size), where each node partitions the input space with a random hyperplane. At query time, only points in the same leaf node as the input vector for all trees are considered candidates. Increasing the number of trees increases accuracy at the expense of index size and a slight speed penalty. This implementation is also for the cosine similarity alone. Extrapolating from results with datasets up to $n = 2$ million in size, this implementation would be capable of calculating pairwise similarities between 10 million input vectors on a single server in approximately one day, which is enough to be useful, considering around 18 million unique users are observed per month. We would like to have experimented with hierarchical k-means combined with the Gaussian similarity method, but ran out of time.

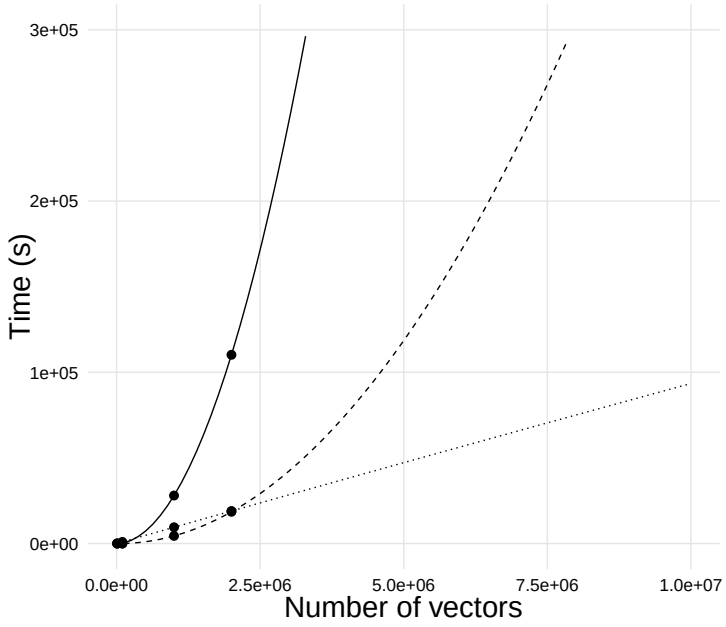


Figure 6.1: Computational results from investigating the 'weak scaling' of the k-nearest neighbour calculation, or the increase in time required for increasing n with constant computational budget. The 'naive' exhaustive method is the solid line, the inverted index method is the dashed line, and the random projection tree method is the dotted line. The first two methods scale with n^2 and the random projection method scales with n.

6.2.2 Calculating eigenvalues of large, sparse matrices

Recall the objective of finding an m-dimensional nonlinear embedding of the user vector data points, given by the eigenvectors of generalised eigenvalue problem $Ly = \lambda Dy$ corresponding to the m smallest non-zero eigenvalues. We now describe the methods we used to solve this and similar eigenproblems in this project. An overview of available eigensolver software is given by Hernandez et al. [56]. Only sparse solvers are applicable due to the size of the matrices involved.

The recently proposed FEAST algorithm [105] was used to solve realizations of the eigenproblem above. FEAST can be parallelized to run on multiple servers, and can be used to find eigenvalues in any part of the eigenspectrum. The multiple shift-invert implicitly restarted Lanczos method also has these properties, and has been shown to have a lower computational cost for finding approximately 10% of the eigenvalues of extremely large matrices in a distributed setting [5], which is primarily due to the computational complexity of the complex arithmetic involved in the FEAST contour integral-based spectral projection method.

We use the FEAST algorithm because it is interesting. For the experiments with large matrices, a FORTRAN implementation made available by the author was used. We also experimented with the implicitly restarted Lanczos method and the Krylov-Schur method using the PARPACK/ARPACK [80] and SLEPc libraries [57] respectively in order to understand the use of these libraries. The PARPACK and SLEPc libraries use MPI for parallel computation and can operate in a distributed-memory environment.

In order to more fully understand the FEAST algorithm, we re-implemented it in MATLAB, shown in 6.2.2. The FEAST algorithm accelerates Rayleigh-Ritz subspace iteration by constructing a functional matrix that projects the matrix of interest onto a subspace that closely approximates the invariant subspace spanned by the eigenvectors inside a given interval on the real line. The

projection matrix is defined as $X_{\mathcal{I}} X_{\mathcal{I}}^H B$ where the columns of $X_{\mathcal{I}}$ are the eigenvectors spanning the subspace, and can be found by approximating a complex contour integral described below.

For the generalised Hermitian eigenvalue problem $A X = B X \Lambda$ the eigenpair is given by (X, Λ). Given an interval $\mathcal{I} = [\lambda_-, \lambda_+]$ on the real line, let $\mathcal{C}$ be the circle centered on the real line and intersecting it at exactly λ_- and λ_+. Let $\pi(\lambda)$ be the complex-valued function defined by the contour integral (in the counter clockwise direction)

$$\pi(\lambda) = \frac{1}{2\pi\iota} \oint_{\mathcal{C}} \frac{1}{z - \lambda}\, dz, \qquad \lambda \neq \mathcal{C}.$$

The Cauchy integral theorem shows that $\pi(\lambda) = 1$ for λ inside the circle $\mathcal{C}$ and $\pi(\lambda) = 0$ for λ outside of $\mathcal{C}$. The idea is that applying π to a matrix whose eigenvalues are Λ corresponds to a spectral projection. Since the GHEP is specified by two matrices, we define one single matrix that serves as a "surrogate". This is

$$\hat{A} \stackrel{\text{def}}{=} X\Lambda X^{-1} = X\Lambda X^H B$$

whose eigenpair is exactly (X, Λ) by design. Assuming just for now that neither λ_- nor λ_+ is an eigenvalue of $\hat{A}$, then the function $\pi(\hat{A})$ is well defined as

$$\pi(\hat{A}) = \frac{1}{2\pi\iota} \oint_{\mathcal{C}} \frac{1}{zI - \hat{A}}\, dz.$$

On the other hand,

$$\pi(\hat{A}) = X\,\pi(\Lambda)X^{-1} = X\,\pi(\Lambda)X^H B = X_{\mathcal{I}} X_{\mathcal{I}}^H B.$$

Hence

$$X_{\mathcal{I}} X_{\mathcal{I}}^H B = \frac{1}{2\pi\iota} \oint_{\mathcal{C}} \frac{1}{zI - \hat{A}}\, dz$$

as long as $\{\lambda_-, \lambda_+\}$ does not intersect with $\hat{A}$'s spectrum. Moreover

$$(zI - \hat{A})^{-1} = (zI - X\Lambda X^H B)^{-1} = [B^{-1}(zB - BX\Lambda X^H B)]^{-1} = (zB - A)^{-1}B.$$

Hence the spectral projection of a set of n-vectors $Y = [y_1, y_2, \ldots, y_p]$ admits an integral representation involving A and B:

$$(X_{\mathcal{I}} X_{\mathcal{I}}^H B)\, Y = \frac{1}{2\pi\iota} \oint_{\mathcal{C}} (zI - \hat{A})^{-1} Y\, dz = \frac{1}{2\pi\iota} \oint_{\mathcal{C}} (zB - A)^{-1} BY\, dz.$$

This integral can be approximated via numerical quadrature using a parallel multi-frontal complex linear system solver such as the MUMPS software [8]. A demonstration implementation of the FEAST algorithm using MATLAB syntax is given in 6.2.2. Note that the MATLAB implementation of Arnoldi factorization for solving the inner eigensystem, which uses a modified version of ARPACK internally.

Listing 6.1: MATLAB implementation of the FEAST algorithm

```
N = 1000;
M = 20;
M0 = ceil(1.5*M);
NI = 8;
A = rand(N); A = A'*A; B = eye(N);
l_min = 10; l_max = 12;
m_eps = eps; e_eps = 1E-10;

% choose M0 random vectors Y
Y = rand(N,M0);

% B-orthonormalize Y via Cholesky Factorization
R = chol(Y'*B*Y); Q = Y/R;

% spectral projection
c = (l_max + l_min)/2;
r = (l_max - l_min)/2;
[x,w] = gl(NI,eps);
z = c + r*exp(1i*(pi/2)*(1 + x));

trace_old = 1
while (trace_residual > 1e-8)
   Y = zeros(N,M0);
   for I = 1 : NI
      PROD = B*Q;
      TEMP = (z(I)*B - A)\PROD;
      Y = Y + (w(I)/2)*real(r*exp(1i*(pi/2)*(1 + x(I)))*TEMP);
   end

   AH = Y'*A*Y;
   BH = Y'*B*Y;
   [X,L] = eig(AH,BH);
   e = diag(L);
   X = scaled(X);
   flags = logical((e > l_min).*(e < l_max));
   total = sum(flags);
   Q = Y*X; Q = scaled(Q);
   e_matched = e(flags); Q_matched = Q(:,flags);
   trace_new = sum(e_matched);

   trace_residual = abs(trace_new - trace_old);
   trace_old = trace_new;
   sum_residual = 0;
   for I = 1 : total
      vector_residual = norm(A*Q_matched(:,I) - e_matched(I)*B*Q_matched(:,I));
      sum_residual = sum_residual + vector_residual;
      sum_residual = sum_residual/norm(AH);
   end
end
```

The complexity of the eigendecomposition is dicussed by Saad et al. [113] and Pan and Chen [98]. This algorithm does not significantly improve on the complexity of the multiple shift-invert implicitly restarted Lanczos procedure. To obtain k extremal eigenvalues of a matrix the Arnoldi method has computational cost of $(O(ma^3 + (O(nma) + O(nk)) * O(ma - m)) * (restarts)$ and a memory cost of $O(nk) + O(nm)$ memory where m is the Arnoldi length. For $n = 20^6$ and $m = 200$ this is a memory requirement of approximately 102 GB.

6.2.3 Using the spectral embedding as generalised linear model features

We trained a generalised linear model to predict the target of whether a user would interact with a given campaign, using the original user vectors (the user-website feature space) and the embedded user vectors as features. We used 16 of the 96 campaigns considered in the previous chapter, with the following steps

1. Compute the user feature space for each campaign along with the target or interaction vector
2. Calculate a training, validation, and test split

3. Compute the k-nearest neighbour similarity graph or matrix. We used the sparse, naive method from subsection 6.2.1 for the heat kernel metric and the inverted index method for the cosine similarity to ensure the exact nearest neighbours are computed with $k = 10$. We note that there are over 18 million unique users in the training dataset. As the cost to calculate the k-nearest neighbour graph for all 18 million was considered too high, we selected 2 million users from the 16 campaign dataset at random, and carried out the embedding process for these users. A similar number of users remaining were embedded in the same position as their closest neighbour in the original space, as this calculation is $O(n)$, not $O(n^2)$. Even these calculations took around 10 hours to run on a ten-node, 160-core cluster such as the one described above.

4. Form the Laplacian matrix as defined above.

5. Calculate the eigenvectors corresponding to the m smallest non-zero eigenvalues for the Laplacian and normalized Laplacian matrices, and the m eigenvectors corresponding to the m largest eigenvalues for the adjacency matrix using the eigensolver described in . We use $m = 20$. This calculation is also performed using a multi-node cluster and the FEAST software.

6. Use the embedded vector created for each user as a set of additional dimensions for each user of the training data used in chapter chapter 5

7. Re-solve the generalised linear model as described in chapter 5

The log-likelihood results for the test dataset for each of the 16 campaigns tested are shown in Table 6.2. There is a marginal improvement in log likelihood with the extra features for each campaign, but the increase in the number of model parameters and the significant effort required to compute the extra features dwarfs this improvement, making this method not practically useful.

Campaign	Original Features	Normalized Laplacian Features	Percentage improvement
1	-2,511	-2,506	0.20
2	-4,207	-4,204	0.07
3	-13,257	-13,242	0.11
4	-1,097	-1,082	1.37
5	-1,118	-1,094	2.15
6	-1,721	-1,685	2.09
7	-977	-956	2.15
8	-18,824	-18,811	0.07
9	-25,050	-25,030	0.08
10	-3,867	-3,863	0.10
11	-2,300	-2,297	0.13
12	-1,727	-1,725	0.12
13	-1,370	-1,359	0.80
14	-2,429	-2,421	0.33
15	-1,152	-1,143	0.78
16	-1,807	-1,802	0.28

Table 6.2: Log-likelihood improvement for 16 campaigns

We consider two possible reason for why the features are uninformative. First, the number of website interactions observed for each user increases over time. The mean L_0 norm for the

user vectors is different to the majority of user L_0 norms due to the approximately exponential distribution of the number of website interactions per user. However, all user vectors are embedded in a space of fixed dimension. Information is discarded for vectors with a large L_0 norm, and information is imputed for vectors with a small L_0 norm. For example, if a space containing a set of vectors with L_0 norms equal to 5 and a set of vectors with L_0 norms equal to 100 is embedded into a 10-dimensional space, one set of vectors is reduced in the inherent dimension and one set is increased in inherent dimension, either discarding or imputing information. The effect of this variation in position in the original space for otherwise identical users, depending on the time for which they have been observed, is likely to obscure any signal contained in the user position in the embedded space.

Second, the sparse k-nearest neighbours graph construction method results in some users (nodes) having a higher degree distribution than others. The node degree distribution is superficially similar to a power-law distribution, with a minimum degree of 10 and a maximum degree of around 600. The graph has multi-scale structure, in that the typical distance between two randomly chosen nodes increases with the logarithm of the number of nodes in the network. Mihail and Papadimitriou [84] and [51] demonstrate that the largest eigenvalues of the adjacency matrix of a graph with a power-law degree distribution also follow a power-law distribution, and that these eigenvectors express the neighbourhood of the high-degree nodes, rather than interesting clusters, or the manifold structure. The degree to which the eigenvectors of the unnormalised Laplacian matrix used above reflect manifold structure is also influenced by the degree distribution. Nadler and Galun [90] and Zelnik-Manor and Perona [134] also describe the limitations of spectral methods in the presence of noise and multi-scale data.

We also attempted the embedding process using the m largest eigenvalues of the normalised Laplacian matrix given by

$$L = D^{-1/2} L D^{-1/2} \tag{6.16}$$

and the adjacency matrix S. There is significant justification that projections using these matrices offers better performance than the unnormalized Laplacian matrix. However, we did not find significantly better results than those shown for the Laplacian matrix above. We note that a spectral method based on a non-backtracking transition matrix is presented in [71], which appears to offer better performance than the Laplacian or adjacency matrices, but we did not have time to investigate this method.

We also note that even if this method had increased predictive performance over the 'base' generalised linear model new information about existing users is continuously arriving. As user-website interactions are observed over time, each user vector in the original feature space changes, affecting many other user vectors in the embedded space. This makes it necessary to recompute the computationally expensive embedding often, which is undesirably expensive.

Finally, despite the poor results on the real problem we find above, we note a similar idea proposed by Coates et al. [40], in the context of using single-layer neural networks for unsupervised feature learning. This paper proposes constructing a nonlinear model k-means clustering and a linear model. First the k-means algorithm is used to cluster the data points. Secondly, the data points are represented in a new k-dimensional space where each dimension is the distance to one of the k centroids (or by a radial basis function or other exponential function of this distance). Thirdly, a linear model is trained on the new feature vectors. Coates et al. [40] achieve the most

accurate results at the time of publishing on the CIFAR-10 and NORB datasets. This technique is further discussed in [39], and a scalable implementation of this technique was developed by a Google engineer, described in [115].

6.2.4 Clustering users with spectral clustering

Instead of training a linear classifier using the embedded space as above, we now assume that the majority of points in the user-website feature space inhabit regions of higher density on the manifold, separated by regions of lower density. Intuitively, this corresponds to the assumption that users belong to groups with distinctive patterns of behaviour, rather than varying smoothly and continuously across the whole spectrum of possible behaviour. The manifold regularization method of calculating a 'modified' kernel penalizing sharp changes in gradient along the manifold may then be approximated by constraining the function to be piecewise constant across these clusters or regions. Stated another way, we simply cluster the users and look for differences in aggregate campaign interaction behaviour for the clusters. We emphasize again that this process is more about experimenting with these methods rather than any expectation that it will generate useful results in the original problem. We note the similarity between this basic heuristic and graph regularization [17].

We first try spectral clustering, which involves clustering the embedded user vectors found above with a simple clustering algorithm, typically k-means [24]. The term spectral clustering covers a family of methods which vary in the matrix used for the embedding and the implementation of the k-means clustering algorithm. The k-means algorithm partitions a set of real-valued vectors into k sets $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$ so as to minimize the sum of squared distances within each cluster:

$$\underset{\mathcal{S}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{S}_i} ||\mathbf{x}_j - \mu_\mathbf{i}||^2 \tag{6.17}$$

Where μ_i is the centroid of cluster $\mathcal{S}_i$. A simple algorithm for this NP-hard minimisation problem involves selecting k initial centroids for each cluster by some arbitrary method. Every point is then assigned to the nearest centroid. The centroids are then re-calculated and the process repeated. One strategy for choosing centroids is to choose vectors from the data at random, and re-run the algorithm multiple times, choosing the initialization that produces the best clustering. We instead select one centroid vector at random, and repeatedly select the vector closest to orthogonal to the worst-case point already selected as suggested in [92]. Once the initial vectors are selected, many open-source libraries are available for minimizing the objective function. A drawback of this, and many other clustering methods, is the requirement to select k.

We used spectral clustering to cluster random samples of users of size $10^4 - 10^6$. We found that the k-means clustering is always dominated by a single large cluster, with the remainder of the clusters similar sizes but much smaller than the large cluster. Figure 6.2 shows relative cluster sizes for the 10^5 sample; the other samples display similar behaviour.

To explain this result, we note that the eigenvectors of the Laplacian matrix of a graph can also be used to find partitions of a graph in way which minimises the total weight of cut edges while attempting to create partitions of equal size [117], [99], [30]. However, for a graph with non-uniform degree distribution, spectral partitioning methods tend to produce partitions with different degree distributions, usually with one dense connected partition containing nodes of high degree and the

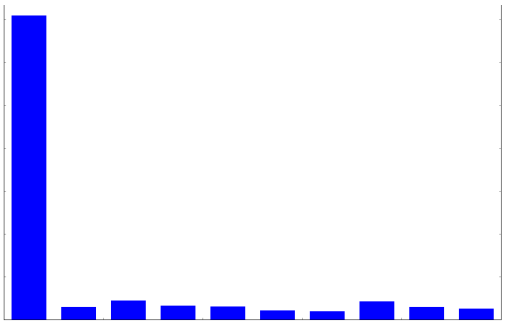


Figure 6.2: Cluster sizes from a sample of 10,000 users clustered using spectral clustering

other with lower-degree nodes that are 'trimmed' away from the first partition [122] For graphs that have a pronounced core-periphery structure, algorithms that minimize the edge cut of a partition split the graph between the core and the periphery. An intuitive description of this core-periphery effect comes from the equivalence between normalized cut and the transition probabilities of the random walk [82]. Lang [72] show that graph cut quality varies inversely with cut balance in small-world graphs. For these graphs, spectral graph partitioning tends to cut away a periphery of low-degree nodes from a central densely connected core. If the nodes forming the boundaries between ground-truth communities are of average degree or above, the 'basic' spectral algorithms have difficulty separating the communities. Abou-Rjeili and Karypis [1] discuss partitioning power-law graphs.

6.3 Community detection in the k-nearest neighbour user graph

Given the shortcomings of spectral clustering in graphs with noise and multi-scale features (examples of graphs for which spectral methods perform well include graphs generated from images or finite element meshes) we consider clustering user vectors using the similarity matrix directly, using community detection methods. Community detection methods look for communities, or clusters, in a graph with a higher intra-group edge density compared to inter-group edge density. There is an exceptional volume of literature on community detection, with hundreds of papers published annually in computer science, social science, physics, and computational biology journals and conferences, much of which is centered on domain application and heuristic variations. Two excellent surveys are given by Fortunato [46] and Schaeffer [114]. As stated at the beginning of this chapter, this investigation is more concerned with experimenting with community detection methods than a belief that the clusters found will exhibit significantly different campaign interaction behaviour.

6.3.0.1 Modularity maximisation

Although modularity maximisation is an older method more recent methods with better performance exist (such as stochastic block models and non-negative matrix factorization based clustering), greedy agglomerative modularity maximisation methods easily scale to graphs with hundreds of millions of nodes, unlike many more performant methods. We therefore investigate two methods for modularity maximisation, a greedy agglomerative method and a method based on simulated annealing.

Clustering the user similarity graph involves assigning each node in the sparse user similarity graph a class that maximises the 'quality' of the graph clustering. The well-known graph modularity

proposed by Girvan and Newman [50] is often used as a measure of quality of the node to community assignment for a graph with associative community structure, and can be expressed by

$$Q\left(\{\sigma\}\right) = \frac{1}{2M} \sum_{i \neq j} \left(A_{ij} - \frac{k_i k_j}{2M} \right) \delta_{\sigma_i, \sigma_j} \tag{6.18}$$

where M is the number of undirected edges in the network, k_i is the degree of node i, A is the adjacency or similarity matrix, and σ_i is the class membership of node i. Finding the vertex to cluster assignment that corresponds to the global maximum of the modularity is an NP-hard problem [27]. However, in practice local minima are practically useful.

One family of algorithms for maximising the modularity includes single-level and multi-level greedy agglomerative algorithms, many of which are derived from a single level greedy algorithm by Clauset et al. [38]. This method begins with every node belonging to its own community, and iteratively merges communities resulting in the greatest modularity increase. Improvements have been made by Blondel et al. [25] and Wakita and Tsurumi [127]. Noack and Rotta [93] developed a hierarchical improvement to the single level algorithm, and a parallel version was later developed by Riedy et al. [111]. Many later papers on the performance characteristics of modularity maximisation and further improvements exist. Given the prevalence of these methods, numerous open-source implementations exist, and we experimented with several, selecting the C++ software by Guillaume Blondel et al. [25] for use in the computational experiments below.

Another approach to modularity maximisation comes from statistical physics [108]. This is based on the equivalence of maximising the modularity to minimising the negative Hamiltonian of a Potts spin glass model [130] constructed with a coupling between every pair of nodes in the lattice. In the Potts model, nodes can have an energetically strongly favourable (ferromagnetic) interaction, or a weakly unfavourable antiferromagnetic interaction. In the graph context, the graph node assignment to one of k communities corresponds to a node's spin assignment in the Potts model lattice. If two nodes are connected, there is an 'energetically favourable' interaction between two nodes, and an 'energetically unfavourable' interaction if the nodes are not linked, both relative to a null model for the graph. With sensible choices for the contribution of existing and non-existing edges to the graph 'energy', the Potts model can be used to find an energetically favourable node spin assignment by minimizing the Hamiltonian

$$\mathcal{H}\left(\{\sigma\}\right) = -\sum_{i \neq j} \left(A_{ij} - \gamma p_{ij} \right) \delta_{\sigma_i, \sigma_j} \tag{6.19}$$

where σ_i is the spin (community) assignment of node i, and p_{ij} represents the probability that a link exists between node i and node j, normalized so that $\sum_{i \neq j} p_{ij} = 2M$. This formulation compares the true distribution of links A with the expected distribution of links under the null model defined by p_{ij}. If $\gamma = 1$ and $p_{ij} = k_i k_j / 2M$ then 6.19 is equivalent to the Newman modularity Reichardt and Bornholdt [108]. The γ parameter allows recovery of community structures of different scales, allowing hierarchical community identification.

A common algorithm used to minimise this quality function or Hamiltonian is simulated annealing, also inspired by statistical physics [67]. The Boltzmann distribution is central to this method,

and is used to represent the probability of particular node spin assignment $\mathbf{x}$ as follows

$$\Pr(\mathbf{x}) \propto \exp\left(\frac{f(\mathbf{x})}{T}\right) \tag{6.20}$$

where T is the computational temperature and f is the Hamiltonian energy function above (6.19). The probability that the node spin assignment is not found in the minimum energy state is therefore strongly dependent on the the value of T. Given an initial node assignment, the simulated annealing algorithm iteratively updates the node spin assignment while the computational temperature is decreased exponentially from a high initial value. At each iteration, the heuristic draws a new spin assignment $\mathbf{x}'$ drawn from a proposal distribution $\mathbf{x} \sim q(\cdot|\mathbf{x})$, and computes the value of the energy function for the current state and the proposed state. Once this is done, the value

$$\alpha = \exp\left(\frac{f(\mathbf{x}) - f(\mathbf{x}')}{T}\right) \tag{6.21}$$

is computed. The algorithm then accepts the proposed state with probability $\min(1, \alpha)$. The algorithm therefore probabilistically decides between moving the system to state $\mathbf{x}'$ or staying in state $\mathbf{x}$ depending on the value of α. The system therefore moves to a lower energy state with probability 1 and moves to a higher energy state with a non-zero probability, enabling moves out of local minima. Typically this iterative process is repeated until the system reaches an energy that is good enough for the application, or until a given computation budget has been exhausted.

We implemented a simulated annealing algorithm to minimise (6.19) for a unipartite, undirected graph for use in the computational experiments described in subsubsection 6.3.0.3. The algorithm is shown in algorithm 6.1. In this algorithm, `initializeNodeSpins` selects an initial spin (cluster) configuration for the graph nodes, and $f(\cdot)$ is the Hamiltonian (6.19). The function `proposeCandidate`, which selects a 'neighbouring' spin configuration $\mathbf{x}'$ to $\mathbf{x}$ by changing the spin assignment for the current node *node* can be arbitrarily specified. We began with a random selection and then change this to selecting the weighted vote of the neighbours of the current node, choosing randomly in the case of a tie.

It is possible to select an optimal number of spin states by re-running the algorithm multiple times and comparing the maximum modularity achieved with the number of spin states, noting that the maximum possible modularity for a given graph increases with the number of spin states, until the the number of spin states equals the number of structural equivalence classes in the network. An optimimum number of spin states can be chosen by evaluating the ratio of the modularity achieved for a given network to the null model for the network and selecting the number of spin states so as to avoid overfitting.

6.3.0.2 Community detection example with the MNIST dataset

In order to illustrate clustering a graph formed from k-nearest neighbour similarity graph of a dataset for which the points lie approximately on a low-dimensional manifold, we consider a simple dataset for which the ground-truth communities are known: the common MNIST digit dataset [74]. This dataset has long been superceded as a demonstration of the capabilities of an algorithm, and is only used here for illustration purposes. The dataset consists of a set of 28×28 pixel greyscale images of the 10 Arabic numerals. Each image is described by a 784-dimensional vector, where each element of the vector is a floating point number representing the pixel intensity.

Input: k_{max}, the maximum number of temperature-decrease iterations
Input: T_{max}, the maximum temperature
Input: c_r, the cooling rate
Input: E_{max}, a target energy (optional)
Output: Node spin assignment $\mathbf{x}$
$\mathbf{x} \leftarrow$ initializeNodeSpins()
$k \leftarrow 0$
$Q_{current} \leftarrow f(\mathbf{x})$
$Q_{new} \leftarrow f(\mathbf{x})$
while $k < k_{max}$ ***and*** $E > E_{max}$ **do**
  $T \leftarrow T_{max}\exp(c_r k)$
  for $node \in G$ **do**
    $\mathbf{x}' \leftarrow$ proposeCandidate$(\mathbf{x}, node)$
    $Q_{new} \leftarrow f(\mathbf{x})$
    if $Q_{new} < Q_{current}$ **then**
      $Q_{current} \leftarrow Q_{new}$
      $\mathbf{x} \leftarrow \mathbf{x}'$
    else
      $\alpha = \exp((f(\mathbf{x}) - f(\mathbf{x}'))/T)$
      if $\alpha > rand()$ **then**
        $\mathbf{x} \leftarrow \mathbf{x}'$
      else
        continue
      end
    end
  end
  $k \leftarrow k + 1$
end
return x

Algorithm 6.1: Modularity maximisation through simulated annealing

We first train two linear classifiers, logistic regression [54] and a single-layer neural network [74] on a set of 50,000 training images and test the accuracy of the classifiers on a set of 10,000 test images. Both of these classifiers achieve a digit classification error of approximately 13%. In order to simulate a context with many unlabelled examples, we then randomly select 10 vectors from the training set as 'labelled' examples and retrain the classifiers above on these examples alone, resulting in significantly reduced classification performance on the test dataset of approximately 30% classification error.

In order to utilize the marginal distribution $\mathcal{P}_X$ of the unlabelled examples, we embed the points using the Laplacian eigenmap method discussed above, cluster the embedded points with $k = 10$, and constrain the classifier to output the same value for each cluster, a very simple graph regularization. Using the 10 labelled examples and the clusters, we achieved 27% classification error, only a marginal improvement on the unregularized classifier performance. However, using the clusters found by modularity maximisation using the simulated annealing method, we achieved 4.8% accuracy using only 10 labelled examples. Figure 6.3 shows a graphical layout of the clustered MNIST graph.

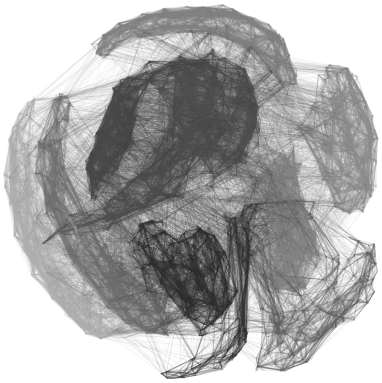


Figure 6.3: MNIST digit dataset clustered by modularity maximisation and laid out with the sfdp algorithm [62]

6.3.0.3 Community detection example with the sparse user similarity graph

We tested both the greedy agglomerative algorithm by Blondel et al. [25] and our simulated annealing implemention for maximising the modularity function of a user-website graph formed using a random sample of 10^6 user vectors. The unclustered user adjacency matrix is shown in Figure 6.4a. Due to the pre-processing methods, the users, or nodes, in the unclustered graph are numbered according to the number of websites seen by each user (the degree distribution of the bipartite user-website graph, distinct from the user similarity graph discussed here). This leads to the observable structure in this matrix. The largest block belongs to users with only 3 website connections. The adjacency matrix re-ordered by the clusters found using the greedy agglomerative algorithm is shown in Figure 6.4c, and community structure is clearly evident. The greedy agglomerative method, for which the number of clusters is not specified, finds a few larger clusters and many much smaller clusters, which common behaviour for this method. The adjacency matrix re-ordered by the clusters found using the simulated annealing algorithm is shown in Figure 6.4b. The number of clusters is specified as 10. Due to the computational cost of this method (the algorithm takes over a week to run sequentially on a single core with the cooling rate used) we tried 10, 20, 50, and 200 clusters, and found 10 clusters to have the highest subjective quality, or rate of increase of the achieved modularity with respect to the number of clusters. As before, community structure is clearly evident.

An important consideration for community detection methods, which is explicit in the Potts model formulation, is the comparison to an appropriate null model for the graph under consideration. Reichardt and White [109] argue that any claim of a practically significant clustering based on a high modularity value must significantly exceed that for a null model based on the graph under study. The authors also show that sparse random graphs can exhibit high maximum modularity values and that exceeding the maximum modularity of a sparse random graph generated by the Erdos-Renyi model [128] by a significant amount is not possible (for a graph of the same degree distribution and number of clusters). The 'discovery' of community structure in a sparse graph should therefore exceed the maximum possible modularity value for a random graph with an equivalent degree distribution.

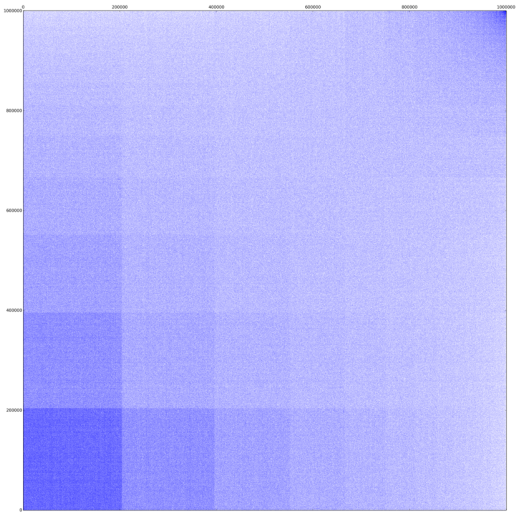
In order to compare the clustering found by the simulated annealing model with a null model, we generated a random graph with an identical number of nodes and degree distribution to the user similarity graph, and ran the same clustering procedure on this graph. The maximum modularity achieved for the user similarity graph with a computational budget of 10,000 temperature-decrease

iterations was 0.74. The maximum modularity achieved for the random graph was 0.11. We note that the maximum possible modularity of a random graph generated using the Erdos-Reyni model with 10^6 nodes and connection probability equal to the average connection probability of the user-website graph has a maximum modularity of 0.233.

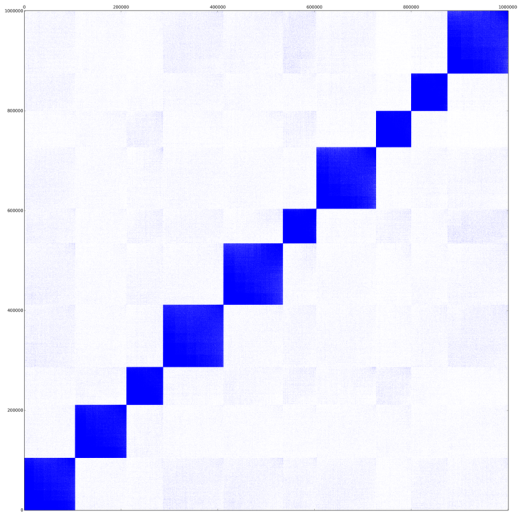
However, despite the significance of the clustering found using the simulated annealing method in a network sense, these clusters do not exhibit significantly different behaviour in terms of campaign interaction probability for the users in each cluster. This is likely due to the clustering being more related to the length of time for which users have been observed (and the number of websites observed for each user) than to the underlying 'true' user position in the feature space, and also to the large cluster size and extremely small effect size we desire to detect.

6.3.0.4 Related work

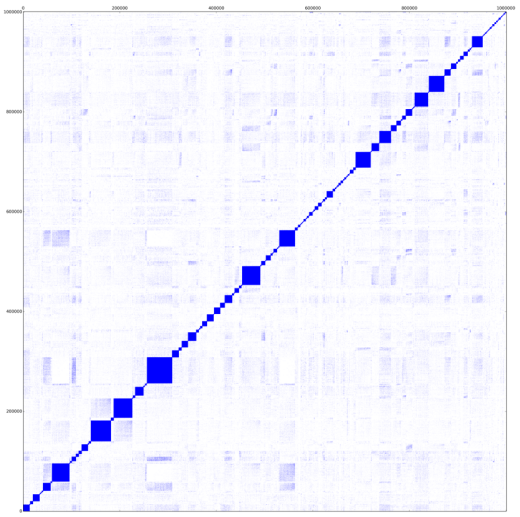
Reichardt and Bornholdt [107] apply modularity maximisation to clustering users of a large online auction platform, using a more advanced implementation of the Hamiltonian-based simulated annealing method discussed above. Reichardt and Bornholdt [107] form a user graph by connecting user vertices with an edge when the users have expressed interest in the same item, conceptually similar to the user graph formed by the k-nearest-neighbour similarity graph described above. In a much more sophisticated investigation than conducted here, the authors then cluster the user graph for varying values of the parameter γ, finding communities at different hierarchical scales, and compare their results favourably with results generated from a null model. The hierarchical communities found in this user graph by the authors exhibit distinct and readily human-interpretable interests and patterns of behaviour as defined by page-view and purchase behaviour on the auction platform.



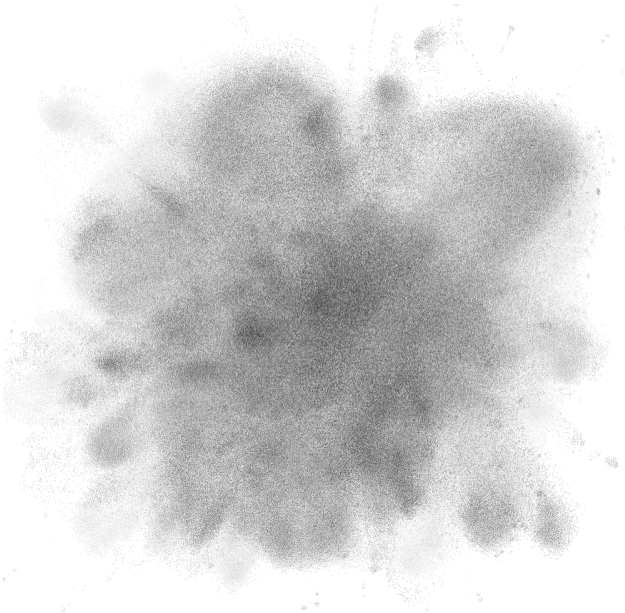
(a) Adjacency matrix for the user similarity graph ordered by preprocessing only



(b) Adjacency matrix for the user similarity graph ordered by the simulated annealing clustering algorithm with $k = 10$



(c) Adjacency matrix for the user similarity graph ordered by the agglomerative greedy clustering algorithm



(d) A user similarity graph with 10^6 nodes, laid out using the **sfdp** algorithm. Assortative community structure is evident.

Figure 6.4: Adjacency matrices and graph layout for a 10-nearest neighbour user similarity graph with 10^6 nodes

6.4 Segmenting the bipartite user-website graph

This section was inspired by considering how to apply community detection to the weighted, bipartite graph formed by directed user-website interactions, rather than the sparse user-user similarity graph. Most community detection methods are applicable to unipartite graphs, and investigating this issue led us to consider network 'role' or block discovery by fitting a stochastic block model. We therefore investigate segmenting users and websites by inferring a degree-corrected stochastic block model over the directed user-website interaction graph.

Degree-corrected stochastic block models explicitly accommodate the variation in vertex degree by giving vertices with higher degree a higher probability of being connected to any other vertex, which avoids the drawbacks of embedding methods described in subsection 6.2.3. After segmenting the users, we investigate differences in advertisement interaction behaviour for the segments. Although the graph clustering is significant relative to an appropriate null model, the differences in advertisement interaction probability are not significant enough to be useful in practice. However, the segments found may be useful in other applications, such as investigating the demographics of users interacting with different types of websites.

6.4.1 Stochastic block models

Graph block models [129] are a family of generative models describing properties of graphs using the concept of structural equivalence between vertices. The block model describes a graph by defining the presence or absence of an edge between classes of structurally equivalent vertices rather than the vertices themselves. Structural equivalence can be generalised to regular equivalence [60], leading to a block structure that 'best matches' the observed edge pattern between vertices in the same class. vertices in the same class are stochastically equivalent as they are exchangeable with respect to the edge probability distribution between the nodes in each class.

The stochastic block model framework allows the expression of a much wider range of interdependencies between network blocks than the independent assortative communities considered in the context of modularity maximisation and similar methods. In particular, stochastic block models can be applied to finding densely connected blocks in bipartite, directed networks - note that this includes blocks where the connectivity is to *each other* rather than densely connected to themselves, termed 'modules' by some authors.

A generative stochastic blockmodel [95] is defined by a directed graph having a specified number of vertices, a number of classes (or blocks, or clusters) q, and a $q \times q$ matrix defining the probability that a vertex of class q_i has a (directed) edge to a vertex of class k_j. This generative model defines a probability distribution over graphs G $\Pr(G|\theta)$ over the set $\mathcal{G}$ of graphs with a defined number of vertices. θ is a parameter vector determining the between-block edge probabilities. Given a realization of a graph, inferring a stochastic block model is the process of determining the parameter vector θ that is most likely to have generated the graph (maximising the posterior likelihood $\Pr(G|\theta)$). The degree-corrected stochastic blockmodel, variants of which are considered from here on, take the degree distribution of each node into account when comparing the between-block edge probability associated with the generative model to the edges observed for a particular node.

Reichardt and White [109] use a similar framework to the stochastic blockmodel framework to define the concept of 'roles' (or blocks) in an m-partite network, rewarding links matching a

generative model of the block relationships and penalize links not matching this generative model, leading to a quality function of the form

$$\mathcal{Q}^* \left(\{\sigma\} \right) = \frac{1}{2} \sum_{r,s}^{q} ||e_{rs} - [e_{rs}]|| \tag{6.22}$$

where e_{rs} and $[e_{rs}]$ are defined as

$$e_{rs} = \frac{1}{M} \sum_{i \neq j} \left(a_{ij} + b_{ij} \right) \, A_{ij} \, \delta_{\sigma_{i,r}} \, \delta_{\sigma_{j,s}} \tag{6.23}$$

$$[e_{rs}] = \frac{1}{M} \sum_{i \neq j} b_{ij} \, \delta_{\sigma_{i,r}} \, \delta_{\sigma_{j,s}} \tag{6.24}$$

where $a_{ij} = 1 - p_{ij}$, $b_{ij} = p_{ij} = k_i^{out} \, k_j^{in} / M$, and as before σ represents the node-class assignment. Note that this formulation compensates for varying node degrees. The e_{rs} variable corresponds to the observed links in the network, and the $[e_{rs}]$ variable corresponds to the number of links expected under the generative model of block relationships. The maximum value of this function occurs when the number of roles equals the number of structural equivalence classes in the network. To avoid over-fitting, the value $\mathcal{Q}^* \left(\{\sigma\} \right)$ of the function is compared with the maximum value

$$\mathcal{Q}_{max} = \frac{1}{M} \sum_{i,j} \alpha_{ij} \, A_{ij} = \frac{1}{M} \left(1 - \frac{k_i^{out} \, k_j^{in}}{M} \right) \, A_{ij}$$

for a given number of roles, and the ratio used to select the optimum number of roles. As noted by the authors Reichardt and White [109], 'assortative community' detection and the modularity objective function emerge as special cases of this framework, where each block connects only to itself, or the $q \times q$ matrix defining relationships between network blocks is diagonal. We note that Reichardt and White [109] develop a method to determine the ideal block structure as a consequence of maximising (6.22) rather than specifying a potentially sub-optimal block structure ahead of time.

A more recent work by Peixoto [100] shows that maximising the posterior likelihood $\Pr(G|\theta))$ of a degree-corrected stochastic block model is equivalent to minimising the entropy function

$$\mathcal{S}_c \approx -E - \sum_k N_k \ln k! - \frac{1}{2} \sum_{rs} e_{rs} \ln \left(\frac{e_{rs}}{e_r e_s} \right) \tag{6.25}$$

where e_{rs} represents the edges between nodes of blocks r and s, $E = \sum_{rs} e_{rs}/2$ is the total number of edges, N_k is the total number of nodes of degree k, and $e_r = \sum_s e_{rs}$ is the total number of half-edges incident on block r. We select this method to infer block structure in a bipartite graph due to the availability of a high-quality software impelementation.

6.4.2 Computational experiments and discussion

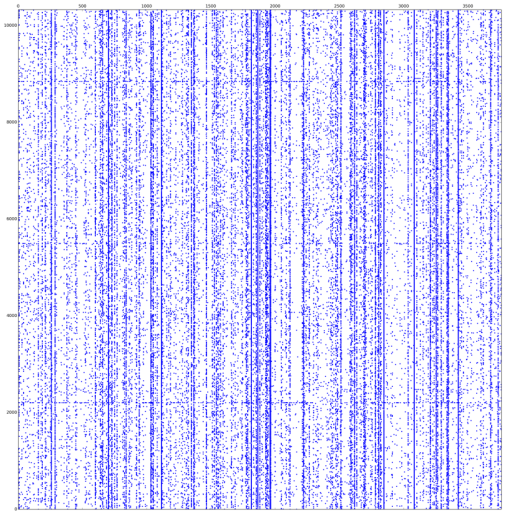
We now consider segmenting users by inferring a degree-corrected stochastic blockmodel for the bipartite graph formed by user-website interactions directly. This avoids the extremely computationally expensive k-nearest neighbour calculation at the expense of introducing a different but potentially equally expensive inference problem. In this setting, the dimensions in the user feature vectors represent integer counts of user-website interactions. These user-website interactions are

used to form a bipartite graph.

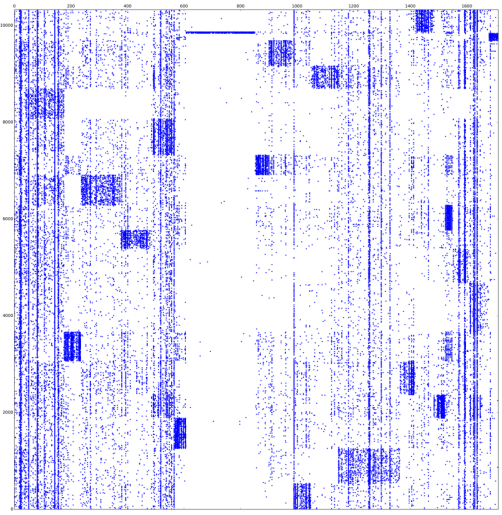
For the numerical experiments, we used the sophisticated software by Peixoto [101] to fit a stochastic blockmodel to biparite user-website networks formed by randomly sampling $10^4, 10^5$, and 10^6 users. This software uses a combination of a greedy heuristic method and a Monte Carlo Markov Chain method to minimise the entropy function (6.25). We defer to [101] for details, noting that greedy heuristic in combination with the Monte Carlo Markov Chain method is the only reason that the library is able to find an acceptably low value for (6.25) in a reasonable time period (even still, fitting a block model to this network took over three weeks).

We achieve a high quality segmentation of the bipartite networks into a block structure, shown in Figure 6.5. As before, we then selected 16 campaigns randomly from the 96 test campaigns, and divided the users exposed to each campaign randomly in half into a train and test set. We then compared the ad interaction rates of the user segments in the training set to those in the test set. However, this was only done with the network containing 10^5 users, and the number of users for each campaign was not large enough to draw significant conclusions, although the results appeared promising for some user blocks. The adjacency matrix with 10^6 nodes shown in Figure 6.5 was fit after the campaign testing dataset was destroyed in order to reclaim space on our laptop; the inference took over 3 weeks of continuous computation, and the campaign datasets hundreds of Gigabytes in size. We note that these clusters may also be useful for other purposes, such as classifying or clustering websites, anomaly detection, and studying user behaviour for purposes not directly related to estimating ad interaction rates.

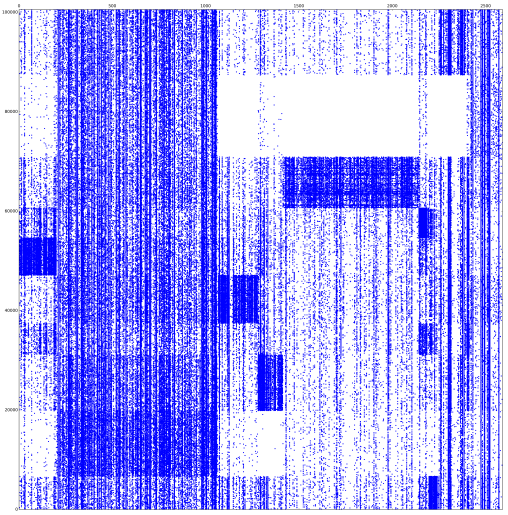
Finally, we note that a linear model constructed using the bipartite user node degree as the only feature is a useful predictor of campaign interaction probability; however, this method does not significantly differentiate between campaigns when tested using the same method as in subsection 5.7.2, merely predicting an overall higher likelihood of user interaction.



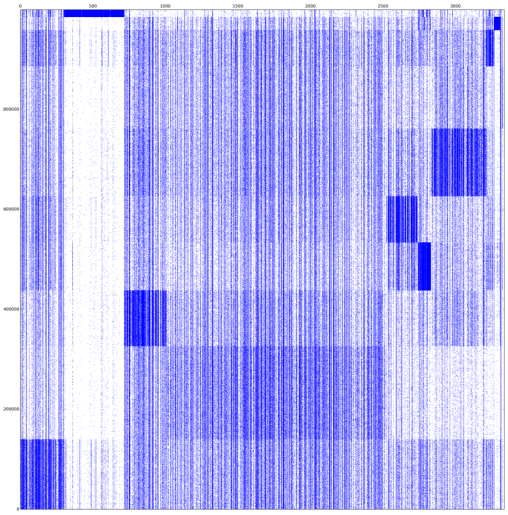
(a) Unordered user-website adjacency matrix with 10^4 rows. Rows are users and columns are websites.



(b) User-website adjacency matrix with 10^4 rows ordered according to inferred block structure.



(c) User-website adjacency matrix with 10^5 rows ordered according to inferred block structure.



(d) User-website adjacency matrix with 10^6 rows ordered according to inferred block structure.

Figure 6.5: Adjacency matrices representing bipartite user-website interaction graphs. Rows correspond to users and columns to websites. In Figure 6.5a rows (users) are ordered randomly and columnns are ordered according to the arbitrary integer website identifier. In Figure 6.5b, Figure 6.5c, and Figure 6.5d the rows and columns are ordered according to the inferred block structure. Note that even for the nearly empty vertical band of websites in Figure 6.5d there is a strongly associated block of around 15,000 users

Summary

In this chapter, we conduct a number of experiment and investigations for their value as learning exercises, rather than direct industrial usefulness. In no particular order, we re-implement the FEAST eigensolver algorithm in MATLAB, and experiment with high-performance distributed eigensolvers in order to embed users into a new feature space using the Laplacian eigenmap technique. We investigate the use of these embedded coordinates as features for the generalised linear model in chapter 5, and comment on the outcome using concepts from spectral graph theory. We investigate fitting degree-corrected stochastic block models to large bipartite user-website networks by applying an existing, but state-of-the-art hierarchical greedy/Monte Carlo Markov chain method. We implement a simulated annealing algorithm for graph community detection via modularity maximisation, and compare this with the common agglomerative greedy approach. Finally we first investigate Laplacian regularized least squares for the same application as the generalised linear model from chapter 5.

Chapter 7

Exploiting real-time bidding market structure and competitor's bids

This chapter contains an overview of the information obtained by analysing all real-time bid auctions involving the exchange considered in this project (as originator or responder) from August to November 2013. We show the expected fluctuations in auction volume due to the variation of user impression volume with time, as well as significant exploitable structure in the bid responses returned by competitors. We also propose using competitor's bid responses for individual auctions to generate additional user features for use in the generalised linear model. Note that while most ad requests originate in Germany, all times are given in UTC/GMT.

7.1 Using external competitor's real-time bids to generate internal user features

When a real-time bid request is sent from the exchange to competitors and other demand-side platforms, the exchange and the recipient platform perform user-matching, where the unique user identifier sent by the exchange is matched with internal tracking data by the recipient. The recipient platform may then use information previously collected about the user to decide how to respond with a bid. Assuming the recipient platform's competence, these bid responses can then be used to generate user features for ad interaction probability modeling within the exchange. For instance, if a certain user repeatedly attracts high bids from Google, then the user will almost certainly have a higher probability of interaction with the exchange's own campaigns than the average. We validated that this approach was possible using the exchange infrastructure and data. However, it was not evaluated at scale as in chapter 5 due to lack of time. We also note that the legality of this method is unknown at the time of writing.

7.2 Exploiting real-time bidding market structure

In order to obtain an overview of the real-time bidding market and advertisement price distributions, we record and analyze real-time bids sent and received by the exchange under consideration over the period Sep-Nov 2013. 38 other firms are represented in this dataset which contains approximately 30 billion events and is 3 Terabytes in size. We processed this dataset using 100 virtual machines in parallel (see subsection 5.3.1 for an explanation of the infrastructure used). Selected major participants in the German online advertising market, including the exchange considered in this project are shown in Table 7.1 along with traffic statistics from July 2013 produced by the research firm ComScore.

The outcome of a real-time bidding auction is theoretically determined by a second price auction, where the highest bidder pays the value of the second highest bid. The optimum strategy in an

	Pageviews (10^9)	Users (10^6)	% of German users	Mean daily users (10^6)
German Internet Audience	104.3	57.9	100	35.8
Google Display Network	43.0	53.6	92.5	22.4
Performance Advertising	3.6	47.6	82.1	12.2
Adscale GMBH	9.4	44.8	77.3	13.1
ValueClick Media EU	2.2	39.2	67.7	8.3
Yahoo! Network Plus	4.3	36.5	63.0	7.2
TradeDoubler	1.4	34.8	60.2	6.0
Vibrant Media	0.6	29.2	50.4	4.2
Microsoft Media Network	1.8	29.0	50.1	5.5
Adconian Media Group	1.2	24.7	42.7	2.9
Specific Media	0.2	16.5	28.5	1.7

Table 7.1: Comscore German internet traffic analysis, July 2013

idealized second price auction is for each party to bid the exact value that winning is worth to them, without attempting to 'game the auction'. For further details refer to Leme and Tardos [76]. In practice, the pure second price auction is often subverted by factors such as soft price floors specified by publishers, which essentially change the second price auction to a first price auction by constraining the winning bidder to pay a price equal to the soft floor.

The data recorded includes the distribution of the maximum bid, the distribution of the second highest bid, and the distribution of the spread between the highest and second highest bids, and the distribution of the remaining bids for 5-minute windows across the 2 month period. The dataset also includes these statistics broken down by the real-time bidding parter and the approximately $10,000$ webpage urls registered with the exchange. This makes it possible to (for instance) identify real-time bidding parters who bid unusually high values for users loading particular webpages. Other ways in which this data is commercially useful include determining the distribution of bids from a particular competitor, predicting the market response to a change in volume by any competitor, and identifying strategies in use by competitors.

7.2.1 Calculating ϵ-accurate quantiles for an unordered, infinite data stream

In order to represent the bid distributions, we calculated a large number of quantiles. The naive method for calculating quantiles is sorting the input and then passing through the sorted input recording the values falling into the positions corresponding to the desired quantiles. For large datasets distributed across multiple servers, this is costly in both computation time and in network data transfer due to the distributed sort. A desirable algorithm for this problem does not require sorted input.

It is possible to very accurately calculate the quantiles of an infinite, randomly ordered data stream in practically-bounded memory using the algorithm proposed by Munro and Paterson [88] and extended by Manku et al. [79]. Munro and Paterson [88] show that it is possible to calculate the k-th highest value from an unordered, potentially infinite input stream with probability proportional to the limited storage used, and provide bounds on the relationship between accuracy and storage. In particular, they show that there is a probability $p > 0$ such that any one-pass algorithm which finds the median of a dataset of size N with probability of failure p requires $O(\sqrt{N})$ storage. Their algorithm maintains b buffers of size k with a weight w and level l. The buffers are initially empty. New elements from the stream are used to populate an empty buffer, assigned a level of zero. When every buffer is full, two buffers with the lowest levels are merged by sorting both buffers contiguously

and selecting every second value. The level of the single output buffer is then incremented by one. New values from the stream are then inserted into a new empty buffer of level zero created following the merge operation. When the input stream is exhausted (or at a given point in time), a weight of 2^l is assigned to every element of a buffer of level $l \in 0, 1, ...L$. The elements of all buffers are then sorted contiguously, and the desired quantiles are selected from this weighted, ordered list.

Manku et al. [79] improve this algorithm by modifying the tree structure of the buffer merges. They further show that by randomly sampling the input stream it is possible to calculating ϵ-approximate quantiles with a probability of success of $1 - p$ with storage requirements of $O\left(\epsilon^{-1} \log^2 \epsilon^{-1} + e^{-1} \log^2 \log p^{-1}\right)$ space, notably independent of the dataset size, allowing continuous computation of the e-approximate quantiles of an infinite stream.

This allows the quantile computation to be performed in a somewhat efficient manner using the mapreduce software framework. The map processes read each real-time bid response in parallel and output a constant value as the key and the bid price as the value. Only one reducer process is created, which takes as input the unordered stream of (key, value) pairs output by the map processes (which is assumed to be randomly ordered with respect to the bid prices). The reducer process maintains the buffers described in subsection 7.2.1 in memory, and continuously discards values with buffer merges. Without the use of the Munro-Paterson algorithm, a sorted copy of the entire dataset would need to be created on disk before streaming to the reducer, greatly increasing the time required for computation. We used an open-source implementation of this method originally developed by Linkedin to compute the results shown below.

7.2.2 Structure in real-time bid market volume and pricing

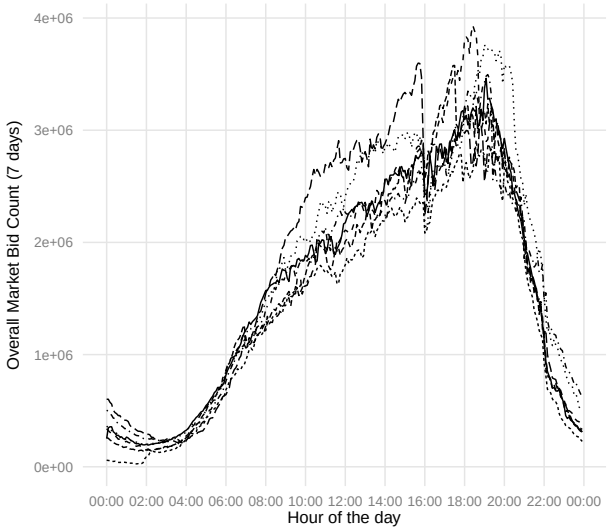
The expectation going into this analysis was that the bid distribution would vary with time of day (as the number of user impressions and purchasing intent varies) with higher demand and prices expected between 6-10pm in the evenings. This behaviour was expected to result from similar pricing models operated by a number of equal competitors in an efficient market. Further, we expected the bid spread (the difference between the largest and second largest bid) to be on average a small percentage of the largest bid.

Figure 7.1 shows that the total number of ad impressions (and real-time bid auctions) varies on a 24-hour night and day cycle as expected. What was not expected is that the highest prices are bid during the early morning, around 0600 hours UTC, with the most bid percentiles including the 95^{th} percentile decreasing from this point onward over the 24 hour day. A possible explanation for this behaviour is that some kind of daily budget is being renewed overnight, and the competitors' decision engines take full advantage of the renewed budget early in the day, decreasing the available budget over the day, resulting in reduced bid prices later in the day.

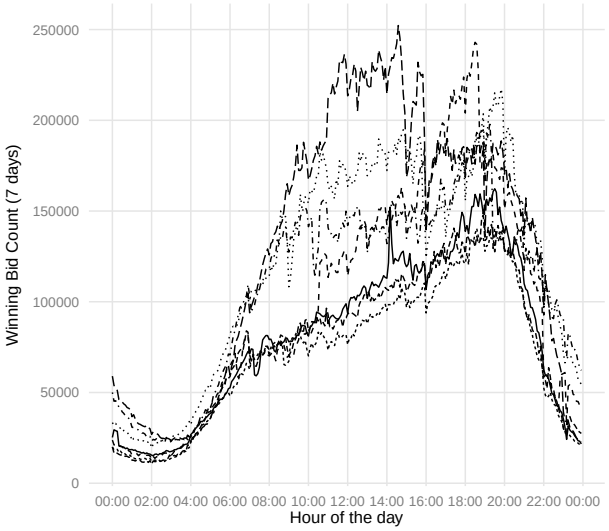
We also note that there are sharp 'steps' in the overall market cumulative bid distribution that persist for hours at a time, as seen in Figure 7.3. One possible explanation for these levels is fixed parameters in competitor's pricing algorithm that result in a 'ceiling' for bids made by that algorithm in certain contexts. A second possible explanation is soft auction floors specified by publishers, where a publisher specifies a minimum desired price, and if the winning bid is lower than this price, the winning bidder is constrained to pay the soft floor price. Note that in this situation, the real-time bid auction is no longer a second price auction.

Figure 7.4 shows cumulative distribution functions for real-time bids received from four of the exchange's largest real-time bidding competitors over 20 minute periods. The maximum bid is over

$30 in each case except for mbr-targeting. All competitors have a similar number of bids, on the order of 10^6 in the 20 minute period. Note the different bidding behaviour for each competitor, and the higher bid values between an instinctively less optimal time of day (early morning) and an instinctively more optimal time of day (early evening).

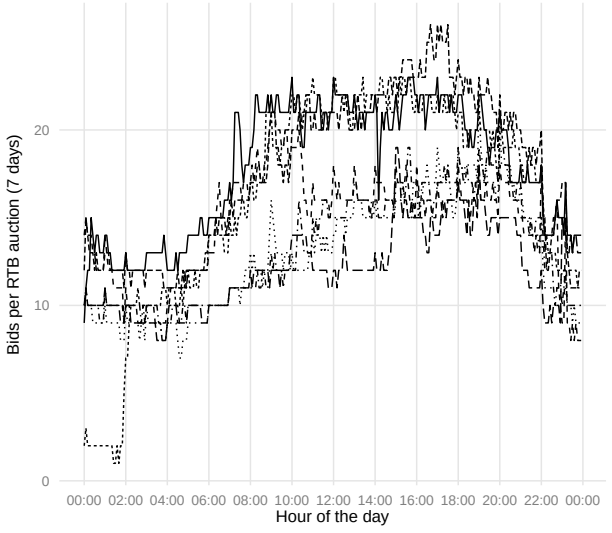


(a) Number of real-time bids recorded by the exchange aggregated over 5 minute intervals, plotted over 24 hours. Each line corresponds to a consecutive day, and 7 days are shown.

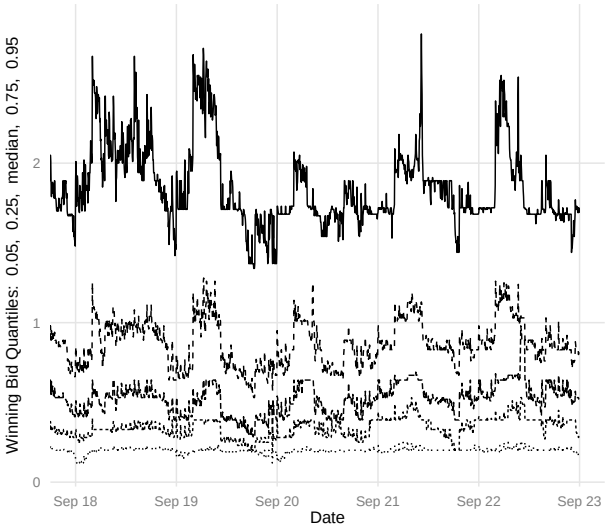


(b) Number of winning real-time bids recorded by the exchange aggregated over 5 minute intervals, plotted over 24 hours. Each line corresponds to a consecutive day, and 7 days are shown.

Figure 7.1: Volumes of real-time bids recorded in September 2013

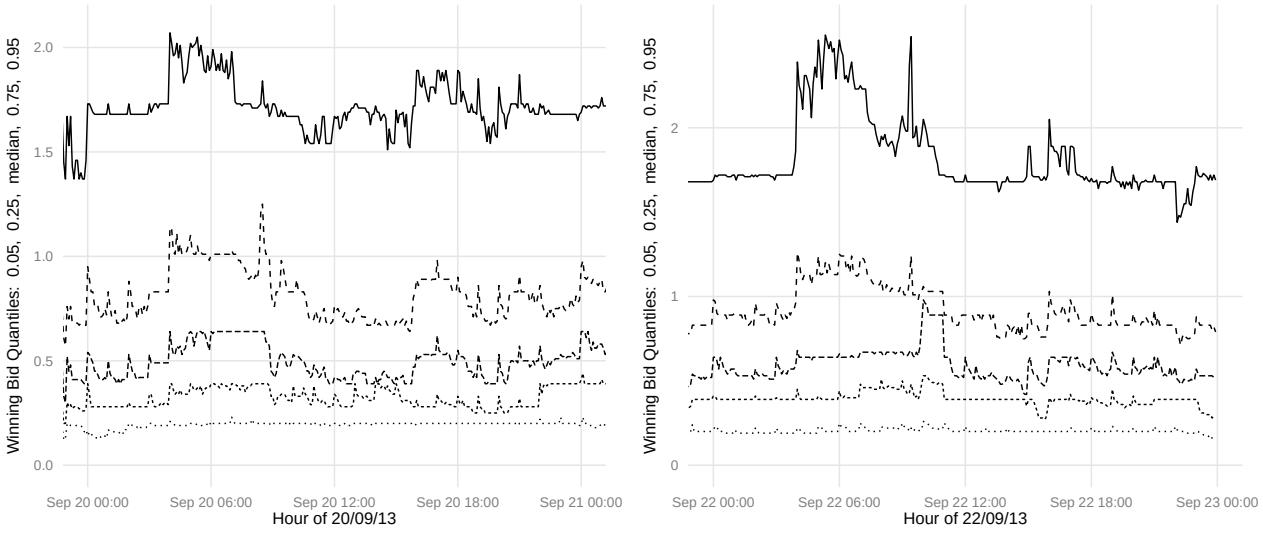


(a) Quantiles for the number of real-time bids per ad impression recorded by the exchange aggregated over 5 minute intervals, plotted over 24 hours



(b) Quantiles for all bids received by the exchange aggregated over 5 minute periods, plotted over several days. The following quantiles are shown: 0.05, 0.25, 0.5, 0.75, and 0.95. Note the defined thresholds, which are believed to indicate thresholds in competitor's bidding algorithms

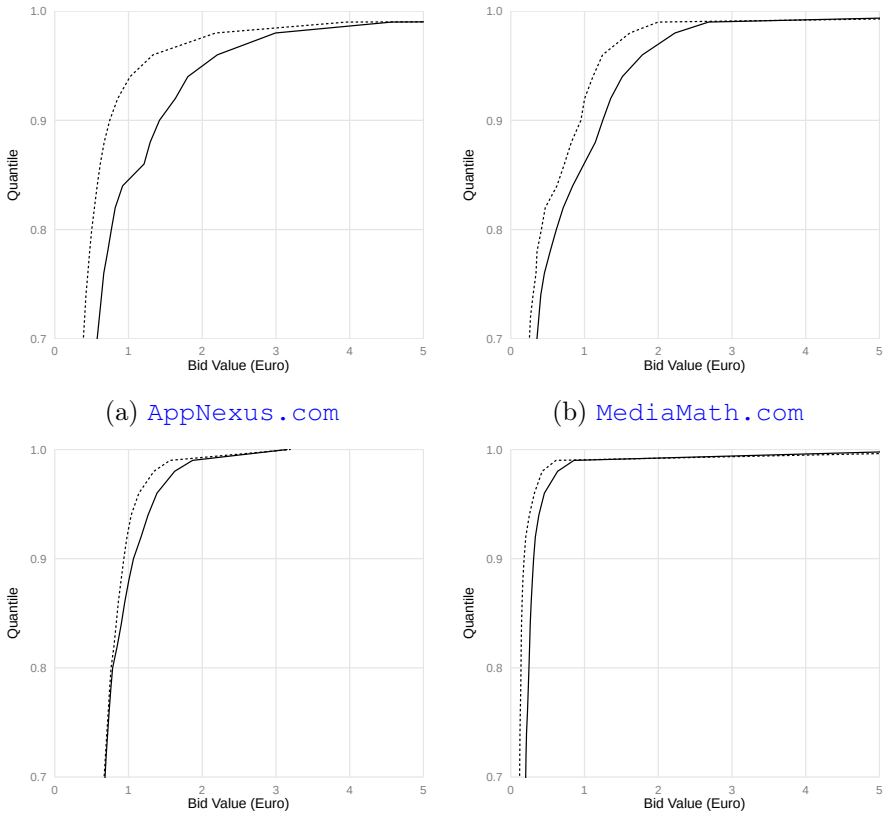
Figure 7.2: Bids per auction and bid quantiles for real-time bids recorded in September 2013



(a) Quantiles over 24 hours on 20/09/2013

(b) Quantiles over 24 hours on 22/09/2013

Figure 7.3: Quantiles for all bids received over 24 hours for the 20/09/2013 and 22/09/2013, split into 5 minute bins. The following quantiles are shown: 0.05, 0.25, 0.5, 0.75, and 0.95. Note the defined thresholds, which are believed to indicate thresholds in competitor's bidding algorithms



(a) AppNexus.com

(b) MediaMath.com

(c) mbr-targeting.com

(d) Turn.com

Figure 7.4: Cumulative density functions for real-time bids received from four of the exchange's real-time bidding competitors with the highest traffic. The dashed lines correspond to a period of 20 minutes from 12:00:00 22/09/2013. The solid lines correspond to a 20 minute period from 06:00:00 22/09/2013.

Summary

In this section, we present an overview of a quantitative investigation into the German real-time bidding market in September and November 2013, covering a substantial fraction of the market. The processed data from which the examples presented are drawn is broken down by real-time bidding competitor, publisher website, and 5-minute period over more than two months. Much more detailed analysis is possible and useful, but is beyond the scope of this document.

References

[1] Abou-Rjeili, A. and Karypis, G. (2006), "Multilevel algorithms for partitioning power-law graphs," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, IEEE, pp. 10–pp.

[2] Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. (2011), "A reliable effective terascale linear learning system," *arXiv preprint arXiv:1110.4198*.

[3] Agarwal, D., Agrawal, R., Khanna, R., and Kota, N. (2010), "Estimating rates of rare events with multiple hierarchies through scalable log-linear models," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 213–222.

[4] Agarwal, D. and Chen, B.-C. (2009), "Regression-based latent factor models," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 19–28.

[5] Aktulga, H. M., Lin, L., Haine, C., Ng, E. G., and Yang, C. (2014), "Parallel eigenvalue calculation based on multiple shift–invert Lanczos and contour integral based spectral projection method," *Parallel Computing*.

[6] Alabduljalil, M., Tang, X., and Yang, T. (2013), "Cache-conscious performance optimization for similarity search," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, ACM, pp. 713–722.

[7] Alabduljalil, M. A., Tang, X., and Yang, T. (2013), "Optimizing parallel algorithms for all pairs similarity search," in *Proceedings of the sixth ACM international conference on Web search and data mining*, ACM, pp. 203–212.

[8] Amestoy, P. R., Duff, I. S., LExcellent, J.-Y., and Koster, J. (2001), "MUMPS: a general purpose distributed memory sparse solver," in *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, Springer, pp. 121–130.

[9] Andoni, A., Indyk, P., Nguyen, H. L., and Razenshteyn, I. (2014), "Beyond Locality-Sensitive Hashing." in *SODA*, SIAM, pp. 1018–1028.

[10] Andrew, G. and Gao, J. (2007), "Scalable training of L 1-regularized log-linear models," in *Proceedings of the 24th international conference on Machine learning*, ACM, pp. 33–40.

[11] Anil, R., Owen, S., Dunning, T., and Friedman, E. (2010), *Mahout in Action*, Manning Publications Co. Sound View Ct. 3B Greenwich, CT 06830.

[12] Awekar, A. and Samatova, N. F. (2009), "Fast matching for all pairs similarity search," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, IEEE Computer Society, pp. 295–300.

[13] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. (1997), "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," in *Modern Software Tools in Scientific Computing*, eds. Arge, E., Bruaset, A. M., and Langtangen, H. P., Birkhäuser Press, pp. 163–202.

[14] Bawa, M., Condie, T., and Ganesan, P. (2005), "LSH forest: self-tuning indexes for similarity search," in *Proceedings of the 14th international conference on World Wide Web*, ACM, pp. 651–660.

[15] Bayardo, R. J., Ma, Y., and Srikant, R. (2007), "Scaling up all pairs similarity search," in *Proceedings of the 16th international conference on World Wide Web*, ACM, pp. 131–140.

[16] Bayer, R. and McCreight, E. (2002), *Organization and maintenance of large ordered indexes*, Springer.

[17] Belkin, M., Matveeva, I., and Niyogi, P. (2004), "Regularization and semi-supervised learning on large graphs," in *Learning theory*, Springer, pp. 624–638.

[18] Belkin, M. and Niyogi, P. (2001), "Laplacian eigenmaps and spectral techniques for embedding and clustering." in *NIPS*, vol. 14, pp. 585–591.

[19] — (2003), "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, 15, 1373–1396.

[20] Belkin, M., Niyogi, P., and Sindhwani, V. (2006), "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *The Journal of Machine Learning Research*, 7, 2399–2434.

[21] Bernhardsson, E. (2013), "annoy, Approximate Nearest Neighbors Something Something," https://github.com/spotify/annoy.

[22] Beygelzimer, A., Kakade, S., and Langford, J. (2006), "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*, ACM, pp. 97–104.

[23] Billsus, D. and Pazzani, M. J. (1998), "Learning Collaborative Information Filters." in *ICML*, vol. 98, pp. 46–54.

[24] Bishop, C. M. et al. (2006), *Pattern recognition and machine learning*, vol. 1, springer New York.

[25] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008), "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, 2008, P10008.

[26] Bottou, L. (1998), "Online Algorithms and Stochastic Approximations," in *Online Learning and Neural Networks*, ed. Saad, D., Cambridge, UK: Cambridge University Press, revised, oct 2012.

[27] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2007), "On finding graph clusterings with maximum modularity," in *Graph-Theoretic Concepts in Computer Science*, Springer, pp. 121–132.

[28] Braun, M. and Moe, W. W. (2013), "Online display advertising: Modeling the effects of multiple creatives and individual impression histories," *Marketing Science*, 32, 753–767.

[29] Candes, E. J. and Plan, Y. (2010), "Matrix completion with noise," *Proceedings of the IEEE*, 98, 925–936.

[30] Chan, P. K., Schlag, M. D., and Zien, J. Y. (1994), "Spectral k-way ratio-cut partitioning and clustering," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13, 1088–1096.

[31] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008), "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, 26, 4.

[32] Chapelle, O., Microsoft, E. M., and Linkedin, R. R. (????), "A Simple and scalable response prediction for display advertising," .

[33] Chapelle, O. et al. (2013), "A Simple and scalable response prediction for display advertising," .

[34] Charikar, M. S. (2002), "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ACM, pp. 380–388.

[35] Chen, L., Tsang, I. W., and Xu, D. (2012), "Laplacian embedded regression for scalable manifold regularization," *Neural Networks and Learning Systems, IEEE Transactions on*, 23, 902–915.

[36] Christiansen, C. L. and Morris, C. N. (1997), "Improving the statistical approach to health care provider profiling," *Annals of Internal Medicine*, 127, 764–768.

[37] Ciaramita, M., Murdock, V., and Plachouras, V. (2008), "Online learning from click data for sponsored search," in *Proceedings of the 17th international conference on World Wide Web*, ACM, pp. 227–236.

[38] Clauset, A., Newman, M. E., and Moore, C. (2004), "Finding community structure in very large networks," *Physical review E*, 70, 066111.

[39] Coates, A. and Ng, A. Y. (2012), "Learning feature representations with k-means," in *Neural Networks: Tricks of the Trade*, Springer, pp. 561–580.

[40] Coates, A., Ng, A. Y., and Lee, H. (2011), "An analysis of single-layer networks in unsupervised feature learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 215–223.

[41] Cortes, C. and Mohri, M. (2007), "On transductive regression," *Advances in Neural Information Processing Systems*, 19, 305.

[42] Dalessandro, B., Hook, R., Perlich, C., and Provost, F. (2012), "Evaluating and optimizing online advertising: Forget the click, but there are good proxies," .

[43] De Francisci, G., Lucchese, C., and Baraglia, R. (2010), "Scaling Out All Pairs Similarity Search with MapReduce," *LSDS-IR10*, 25.

[44] Dean, J. and Ghemawat, S. (2008), "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, 51, 107–113.

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

[46] Fortunato, S. (2010), "Community detection in graphs," *Physics Reports*, 486, 75–174.

[47] Fowlkes, C., Belongie, S., Chung, F., and Malik, J. (2004), "Spectral grouping using the Nystrom method," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26, 214–225.

[48] Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013), *Bayesian data analysis*, CRC press.

[49] Gelman, A. and Hill, J. (2006), *Data analysis using regression and multilevel/hierarchical models*, Cambridge University Press.

[50] Girvan, M. and Newman, M. E. (2002), "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, 99, 7821–7826.

[51] Gkantsidis, C., Mihail, M., and Zegura, E. (2003), "Spectral analysis of Internet topologies," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, IEEE, vol. 1, pp. 364–374.

[52] Graepel, T., Candela, J. Q., Borchert, T., and Herbrich, R. (2010), "Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 13–20.

[53] Griffiths, D. (1973), "Maximum likelihood estimation for the beta-binomial distribution and an application to the household distribution of the total number of cases of a disease," *Biometrics*, 637–648.

[54] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., and Tibshirani, R. (2009), *The elements of statistical learning*, vol. 2, Springer.

[55] Hein, M., Audibert, J.-Y., and Von Luxburg, U. (2005), "From graphs to manifolds–weak and strong pointwise consistency of graph laplacians," in *Learning theory*, Springer, pp. 470–485.

[56] Hernandez, V., Roman, J., Tomas, A., and Vidal, V. (2005), "A survey of software for sparse eigenvalue problems," .

[57] Hernandez, V., Roman, J. E., and Vidal, V. (2005), "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Transactions on Mathematical Software (TOMS)*, 31, 351–362.

[58] Hillard, D., Schroedl, S., Manavoglu, E., Raghavan, H., and Leggetter, C. (2010), "Improving ad relevance in sponsored search," in *Proceedings of the third ACM international conference on Web search and data mining*, ACM, pp. 361–370.

[59] Hofmann, T. (1999), "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, pp. 50–57.

[60] Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983), "Stochastic blockmodels: First steps," *Social networks*, 5, 109–137.

[61] Hoyer, P. O. (2002), "Non-negative sparse coding," in *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, IEEE, pp. 557–565.

[62] Hu, Y. (2005), "Efficient, high-quality force-directed graph drawing," *Mathematica Journal*, 10, 37–71.

[63] Hummel, P. and McAfee, P. (2014), "Machine learning in an auction environment," in *Proceedings of the 23rd international conference on World wide web*, International World Wide Web Conferences Steering Committee, pp. 7–18.

[64] Izrailev, S. and Stanley, J. M. (2014), "Machine Learning at Scale," *arXiv preprint arXiv:1402.6076*.

[65] Karampatziakis, N. and Langford, J. (2010), "Online importance weight aware updates," *arXiv preprint arXiv:1011.1576*.

[66] King, G. and Zeng, L. (2001), "Logistic regression in rare events data," *Political analysis*, 9, 137–163.

[67] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983), "Optimization by simmulated annealing," *science*, 220, 671–680.

[68] Koren, Y. and Bell, R. (2011), "Advances in collaborative filtering," in *Recommender Systems Handbook*, Springer, pp. 145–186.

[69] Koren, Y., Bell, R., and Volinsky, C. (2009), "Matrix factorization techniques for recommender systems," *Computer*, 42, 30–37.

[70] Kruschke, J. (2010), *Doing Bayesian data analysis: A tutorial introduction with R*, Academic Press.

[71] Krzakala, F., Moore, C., Mossel, E., Neeman, J., Sly, A., Zdeborová, L., and Zhang, P. (2013), "Spectral redemption in clustering sparse networks," *Proceedings of the National Academy of Sciences*, 110, 20935–20940.

[72] Lang, K. (2005), "Fixing two weaknesses of the spectral method," in *Advances in Neural Information Processing Systems*, pp. 715–722.

[73] Langford, J., Li, L., and Zhang, T. (2009), "Sparse online learning via truncated gradient," in *Advances in neural information processing systems*, pp. 905–912.

[74] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998), "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86, 2278–2324.

[75] Lee, K.-c., Orten, B., Dasdan, A., and Li, W. (2012), "Estimating conversion rate in display advertising from past erformance data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 768–776.

[76] Leme, R. P. and Tardos, E. (2010), "Pure and Bayes-Nash price of anarchy for generalized second price auction," in *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, IEEE, pp. 735–744.

[77] Li, W., Wang, X., Zhang, R., Cui, Y., Mao, J., and Jin, R. (2010), "Exploitation and exploration in a performance based contextual advertising system," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 27–36.

[78] Liu, Y., Pandey, S., Agarwal, D., and Josifovski, V. (2012), "Finding the right consumer: optimizing for conversion in display advertising campaigns," in *Proceedings of the fifth ACM international conference on Web search and data mining*, ACM, pp. 473–482.

[79] Manku, G. S., Rajagopalan, S., and Lindsay, B. G. (1998), "Approximate medians and other quantiles in one pass and with limited memory," in *ACM SIGMOD Record*, ACM, vol. 27, pp. 426–435.

[80] Maschho, K. J. and Sorensen, D. (1996), "A portable implementation of ARPACK for distributed memory parallel architectures," in *Proceedings of the Copper Mountain Conference on Iterative Methods, April*, pp. 9–13.

[81] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. (2013), "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 1222–1230.

[82] Meila, M. and Shi, J. (2001), "A random walks view of spectral segmentation," .

[83] Menon, A. K., Chitrapura, K.-P., Garg, S., Agarwal, D., and Kota, N. (2011), "Response prediction using collaborative filtering with hierarchies and side-information," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 141–149.

[84] Mihail, M. and Papadimitriou, C. (2002), "On the eigenvalue power law," in *Randomization and approximation techniques in computer science*, Springer, pp. 254–262.

[85] Morales, J. L. and Nocedal, J. (2011), "Remark on Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, 38, 7.

[86] Muja, M. and Lowe, D. G. (2009), "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." in *VISAPP (1)*, pp. 331–340.

[87] — (2014), "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36.

[88] Munro, J. I. and Paterson, M. S. (1980), "Selection and sorting with limited storage," *Theoretical computer science*, 12, 315–323.

[89] Muthukrishnan, S. (2009), "Ad exchanges: Research issues," in *Internet and network economics*, Springer, pp. 1–12.

[90] Nadler, B. and Galun, M. (2006), "Fundamental limitations of spectral clustering," in *Advances in Neural Information Processing Systems*, pp. 1017–1024.

[91] Ng, A. Y. (2004), "Feature selection, L 1 vs. L 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, ACM, p. 78.

[92] Ng, A. Y., Jordan, M. I., Weiss, Y., et al. (2002), "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, 2, 849–856.

[93] Noack, A. and Rotta, R. (2009), "Multi-level algorithms for modularity clustering," in *Experimental Algorithms*, Springer, pp. 257–268.

[94] Norouzi, M., Punjani, A., and Fleet, D. (2013), "Fast Exact Search in Hamming Space with Multi-Index Hashing," .

[95] Nowicki, K. and Snijders, T. A. B. (2001), "Estimation and prediction for stochastic blockstructures," *Journal of the American Statistical Association*, 96, 1077–1087.

[96] Owen, A. B. (2007), "Infinitely imbalanced logistic regression," *The Journal of Machine Learning Research*, 8, 761–773.

[97] ODonnell, R., Wu, Y., and Zhou, Y. (2014), "Optimal lower bounds for locality-sensitive hashing (except when q is tiny)," *ACM Transactions on Computation Theory (TOCT)*, 6, 5.

[98] Pan, V. Y. and Chen, Z. Q. (1999), "The complexity of the matrix eigenproblem," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, ACM, pp. 507–516.

[99] Papadimitriou, C. H. and Steiglitz, K. (1998), "6.1 The Max-Flow, Min-Cut Theorem," *Combinatorial Optimization: Algorithms and Complexity. Dover*, 120–128.

[100] Peixoto, T. P. (2012), "Entropy of stochastic blockmodel ensembles," *Physical Review E*, 85, 056122.

[101] — (2014), "Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models," *Physical Review E*, 89, 012804.

[102] Perlich, C., Dalessandro, B., Hook, R., Stitelman, O., Raeder, T., and Provost, F. (2012), "Bid optimizing and inventory scoring in targeted online advertising," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 804–812.

[103] Perlich, C., Dalessandro, B., Raeder, T., Stitelman, O., and Provost, F. (2014), "Machine learning for targeted display advertising: Transfer learning in action," *Machine Learning*, 95, 103–127.

[104] Plummer, M. et al. (2003), "JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling," in *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003). March*, pp. 20–22.

[105] Polizzi, E. (2009), "The FEAST eigenvalue solver," *URL http://www. ecs. umass. edu/˜ polizzi/feast*.

[106] Řehůřek, R. and Sojka, P. (2010), "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, pp. 45–50, http://is.muni.cz/publication/884893/en.

[107] Reichardt, J. and Bornholdt, S. (2005), "Ebay users form stable groups of common interest," *arXiv preprint physics/0503138*.

[108] — (2006), "Statistical mechanics of community detection," *Physical Review E*, 74, 016110.

[109] Reichardt, J. and White, D. R. (2007), "Role models for complex networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, 60, 217–224.

[110] Richardson, M., Dominowska, E., and Ragno, R. (2007), "Predicting clicks: estimating the click-through rate for new ads," in *Proceedings of the 16th international conference on World Wide Web*, ACM, pp. 521–530.

[111] Riedy, E. J., Meyerhenke, H., Ediger, D., and Bader, D. A. (2012), "Parallel community detection for massive graphs," in *Parallel Processing and Applied Mathematics*, Springer, pp. 286–296.

[112] Ross, S., Mineiro, P., and Langford, J. (2013), "Normalized online learning," *arXiv preprint arXiv:1305.6646*.

[113] Saad, Y., Stathopoulos, A., Chelikowsky, J., Wu, K., and Öğüt, S. (1996), "Solution of large eigenvalue problems in electronic structure calculations," *BIT Numerical Mathematics*, 36, 563–578.

[114] Schaeffer, S. E. (2007), "Graph clustering," *Computer Science Review*, 1, 27–64.

[115] Sculley, D. (2010), "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*, ACM, pp. 1177–1178.

[116] Sculley, D., Otey, M. E., Pohl, M., Spitznagel, B., Hainsworth, J., and Zhou, Y. (2011), "Detecting adversarial advertisements in the wild," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 274–282.

[117] Shi, J. and Malik, J. (2000), "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22, 888–905.

[118] Silpa-Anan, C. and Hartley, R. (2008), "Optimised KD-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, pp. 1–8.

[119] Sindhwani, V., Niyogi, P., and Belkin, M. (2005), "Beyond the point cloud: from transductive to semi-supervised learning," in *Proceedings of the 22nd international conference on Machine learning*, ACM, pp. 824–831.

[120] Sindhwani, V., Niyogi, P., Belkin, M., and Keerthi, S. (2005), "Linear manifold regularization for large scale semi-supervised learning," in *Proc. of the 22nd ICML Workshop on Learning with Partially Classified Training Data*, vol. 28.

[121] Song, Y., Chen, W.-Y., Bai, H., Lin, C.-J., and Chang, E. Y. (2008), "Parallel spectral clustering," in *Machine Learning and Knowledge Discovery in Databases*, Springer, pp. 374–389.

[122] Spielmat, D. A. and Teng, S.-H. (1996), "Spectral partitioning works: Planar graphs and finite element meshes," in *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, IEEE, pp. 96–105.

[123] Sundaresan, N. (2011), "Recommender systems at the long tail," in *Proceedings of the fifth ACM conference on Recommender systems*, ACM, pp. 1–6.

[124] Sussman, D., Tang, M., and Priebe, C. (2013), "Consistent latent position estimation and vertex classification for random dot product graphs," .

[125] Tsang, I. W. and Kwok, J. T. (2006), "Large-scale sparsified manifold regularization," in *Advances in Neural Information Processing Systems*, pp. 1401–1408.

[126] Vapnik, V. N. and Vapnik, V. (1998), *Statistical learning theory*, vol. 2, Wiley New York.

[127] Wakita, K. and Tsurumi, T. (2007), "Finding community structure in mega-scale social networks:[extended abstract]," in *Proceedings of the 16th international conference on World Wide Web*, ACM, pp. 1275–1276.

[128] West, D. B. et al. (2001), *Introduction to graph theory*, vol. 2, Prentice hall Upper Saddle River.

[129] White, H. C., Boorman, S. A., and Breiger, R. L. (1976), "Social structure from multiple networks. I. Blockmodels of roles and positions," *American journal of sociology*, 730–780.

[130] Wu, F.-Y. (1982), "The potts model," *Reviews of modern physics*, 54, 235.

[131] Wu, K.-W., Ferng, C.-S., Ho, C.-H., Liang, A.-C., Huang, C.-H., Shen, W.-Y., Jiang, J.-Y., Yang, M.-H., Lin, T.-W., Lee, C.-P., et al. (2012), "A two-stage ensemble of diverse models for advertisement ranking in KDD Cup 2012," .

[132] Yianilos, P. N. (1993), "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, pp. 311–321.

[133] Zadeh, R. B. and Goel, A. (2013), "Dimension independent similarity computation," *The Journal of Machine Learning Research*, 14, 1605–1626.

[134] Zelnik-Manor, L. and Perona, P. (2004), "Self-tuning spectral clustering," in *Advances in neural information processing systems*, pp. 1601–1608.

[135] Zhang, L., Zhang, Y., Zhang, D., and Tian, Q. (2013), "Distribution-Aware Locality Sensitive Hashing," in *Advances in Multimedia Modeling*, Springer, pp. 395–406.

[136] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. (2010), "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, pp. 2595–2603.