Predictive Crawling for Commercial Web Content

Shuguang Han[1], Bernhard Brodowsky[2], Przemek Gajda[2], Sergey Novikov[2], Mike Bendersky[1], Marc Najork[1], Robin Dua[1] and Alexandrin Popescul[1]
1. Google AI, 1600 Amphitheatre Parkway, Mountain View, CA, USA
2. Google Zurich, Brandschenkestrasse 110, 8002 Zurich, Switzerland
hanshuguang,bernhardb,pgajda,sergeyn,bemike,najork,robindua,apopescul@google.com

ABSTRACT

Web crawlers spend significant resources to maintain freshness of their crawled data. This paper describes the optimization of resources to ensure that product prices shown in ads in a context of a shopping sponsored search service are synchronized with current merchant prices. We are able to use the predictability of price changes to build a machine learned system leading to considerable resource savings for both the merchants and the crawler. We describe our solution to technical challenges due to partial observability of price history, feedback loops arising from applying machine learned models, and offers in cold start state. Empirical evaluation over large-scale product crawl data demonstrates the effectiveness of our model and confirms its robustness towards unseen data. We argue that our approach can be applicable in more general data pull settings.

CCS CONCEPTS

• **Information systems → Web crawling**; *E-commerce infrastructure*; *Search engine architectures and scalability*;

KEYWORDS

Predictive Crawling, Product Search, Commercial Content Change Dynamics

ACM Reference Format:
Shuguang Han, Bernhard Brodowsky, Przemek Gajda, Sergey Novikov, Mike Bendersky, Marc Najork, Robin Dua and Alexandrin Popescul. 2019. Predictive Crawling for Commercial Web Content. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313694

1 INTRODUCTION

An important characteristic of web content is the constancy of its change. While the dynamics of change are well understood in the context of generic web pages [2, 4, 5], commercial content (e.g., product listings, e-commerce sites, etc.) dynamics are relatively less explored. We see two major differences between the generic web content and commercial content. First, commercial content tends to have different change dynamics as it is closely related to marketing and economic factors such as supply and demand. Second, commercial content contains richer domain-specific metadata, e.g., merchant and brand, which provides us with more facets to understand its change dynamics. Besides, since there are billions of products advertised and sold online, often from small independent merchants, tracking and predicting of commercial content change at scale is an important but hard research challenge. Thus, in this paper, we focus on the change dynamics for commercial content.

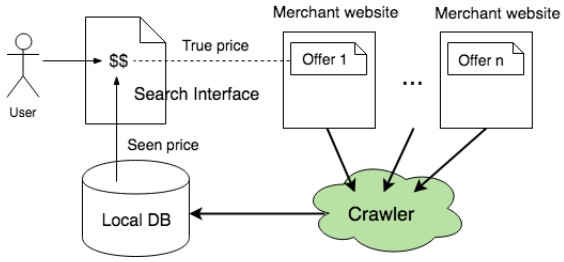

Figure 1: Overall architecture for our product search service.

In particular, we study commercial content change in the context of building an effective product search service. Figure 1 illustrates a high level architecture for such service. Product search results are based on the *product offers* that are listed on *merchants'* websites. These merchants are distributed and independent, and their websites cannot be searched in real-time due to latency constraint. Thus, a local database is used to store local copies of offer attributes. This paper studies the dynamics of price changes since price is one of the most important attributes of online products. The local database is then used by the search interface when users access offer details. The crawler plays a crucial role in our search service, as it attempts to keep the local database fresh. The ultimate goal for the crawler is to ensure that the prices seen by users (from the local database) match the prices shown on merchants' websites.

To achieve this goal, we need to update the local database immediately after an offer changes its price. One potential way is encouraging merchants to push price change information through merchant feeds. Our production system indeed receives such information. This mechanism alone, however, in our experience is insufficient to maintain maximum product offer freshness. Typically, merchant feeds are pushed according to specific schedules, but offers may change attributes regardless of the feed upload schedule. Creating a quality feed and delivering it continuously carries a substantial cost for merchants. Therefore, significant efforts are still required to build an efficient offer crawler.

An alternative solution is to repeatedly crawl merchants' websites for updates. Production crawlers often need to handle billions

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313694

of offers, and scheduling frequent re-crawls for all them is impractical. To deal with that, our production crawler selects a subset of offers based on certain heuristics, and then crawls them periodically using a *Batch Scheduler*. Meanwhile, to compensate for important offers that might not be selected, we include an *InstaCrawl* component to crawl offers immediately after user clicks. Although this brings reasonable database freshness, offer selection heuristics in the *Batch Scheduler* are hard to build, and we overspend substantial resources on *InstaCrawl* because most offers do not change prices.

Previous development of generic web crawlers encounters a similar issue; researchers and practitioners try to solve it by predicting content change and schedule crawls accordingly [19, 24]. Most of existing studies developed their models based on web pages' past change frequencies [12, 13, 16, 20, 25]. There are two issues with this approach. First, it cannot handle the web pages with no or little crawl history, i.e., cold start. Second, it cannot deal with feedback loops — adjustments to the crawl strategy can distort the distribution of change frequency features so that the trained model may not work anymore. Later studies discovered that the content of a web page can help alleviate the cold start problem since similar pages share similar change patterns [28, 30].

Inspired by prior work, in this paper, we apply the predictive crawling approach to the domain of crawling commercial content. Informed by the latest advances in deep learning, we propose a state-of-the-art neural network based method to predict price change at scale. Our method combines both change dynamics as well as content and metadata information based features in a unified framework which enables us to effectively handle cold start and feedback loops of change frequency features.

We further develop new crawling strategies based on the predictive models and examine their effectiveness in the production environment. In this paper, we focus on optimizing *InstaCrawl* because the corresponding offers are more likely to be accessed by users. Particularly, we use the predictive model to reduce crawls for offers that are unlikely to change prices. Our experiments show that the new crawling strategies succeed in saving significant amount of resources. Since we use metadata and content features which are relatively static and independent from crawl history, our methods save resources even in the cold-start scenarios, for offers with little or no prior change history. In this paper, we intend to keep our models generalizable to other crawling components. For example, our price change model can also be applied to the *Batch Scheduler* for selecting more appropriate offers for periodical re-crawls.

To summarize, the main contributions of our work are as follows:

- To motivate our predictive models, we provide a comprehensive, global-scale study of factors that affect online price change – the first such published study to the best of our knowledge (Section 3).
- We develop a state-of-the-art and scalable price change prediction model that combines both price change frequency and metadata features (Section 4), and further demonstrate both its effectiveness and robustness in a cold-start setting (Section 5).
- We utilize the price change prediction model for making crawl decisions, and demonstrate that it can save a significant amount of crawl resources in production setting (Section 6).

2 RELATED WORK

A large body of work addressing freshness in pull systems has been done in the context of web crawlers (e.g. [6, 12, 23]). The topic started attracting significant interest in industry and research in the 1990s together with the rise of web search engines. Crawler optimization to address significant resource demands became an active area of research too.

A number of studies, for example [11, 13, 31], focused on change detection and estimation of frequency of change with sampling based policies. Cho and Garcia-Molina [8, 9] analyzed web content change patterns and proposed several design choices for incremental crawlers. Among them, the authors discovered that web pages changed with different frequency, and the overall freshness of a crawled corpus can be improved by 10%-23% through adjusting download frequency for different pages. Brewington and Cybenko [4] presented statistical analysis of web page modifications, introduced a freshness metric and estimated the download rate to maintain a desired level of freshness. The works [19] and [2] analyzed degree to which change correlates with other page properties, described finer-grained change statistics within different regions in the page, across various types of web pages and user visitation patterns. Tan and Mitra [30] used unsupervised learning to focus crawling within groups of pages that share similar change patterns.

Also, in the context of web page crawling, Radinsky and Bennett [28] extended the prediction model by combining change frequency features with the content of pages and their similarity to other pages. However, the use of change frequency features brings several important issues. Such features are calculated from download history and are sensitive to changes in crawl frequency. Changes to the crawling strategy will significantly affect the distribution of crawl history features, whereas the goal for change prediction is to apply such a model to reduce unnecessary crawling. Radinsky and Bennett's model can potentially avoid such issues since it also considers the content of web pages. However, the authors did not examine the performance after excluding the change frequency features. Also, they assume a full observation of crawl history when computing the change frequency which are unavailable most of the time.

Having a reliable prediction of when a web page will change is not enough for building an effective crawler. A key component in addition to estimating change probability is establishing an update policy. Optimality and scalability of such policies for various metrics under different assumptions is a subject of a number of studies. A recent work [3] presented a tractable near-optimal randomized strategy that can be computed in near-linear time. Eckstein et al. [18] studied the monitoring of web page changes under politeness constraint. Cho and Garcia-Molina [10] introduced freshness and age metrics and proposed several crawler policies assuming that change frequency follows a Poisson distribution. Also under the Poisson assumption, Coffman et al. [15] studied crawler optimality and proved an optimal formula for page refresh frequency, and Grimes et al. [20] studied optimality of a combined cost model for staleness and crawl resources when estimating refresh rates. We show, in this paper, that prices are not equally likely to change across different times of day and days of the week, thus deviating from the Poisson model. Aligned with our findings, Calzarossa and Tesser [5] analyzed temporal change patterns for news websites

and showed that the changes are time dependent with significant hourly and daily fluctuations.

The work from Olston and Pandey [25] addressed crawler optimality which focused on persistent content and attempted avoiding ephemeral content likely to be obsolete by the time it reaches the index. Lefortier et al. [21], on the other hand, studied situations when engines do want to present fresh results even over ephemeral content. Wolf et al. [32] studied crawling strategies prioritized to refresh content that is more likely to surface as a result of a user query. Pandey et al. [27] presented a probabilistic change behaviour model, resource allocation and scheduling mechanism to answer continuous queries over dynamic web content and describe its more subtle differences compared to general web crawling.

A related area of research studies policies for refreshing web content caches. For example, Cohen and Kaplan [16] proposed policies for proactive validation of cache content using historic access patterns such as frequency and recency of access.

Overall, existing studies did conduct extensive research on both crawler optimization and content change prediction. However, they mainly focused on the crawling of generic web pages, and, to the best of our knowledge, very few of them studied the crawling for commercial content. Our work is the first to provide a large-scale analysis of the dynamics of commercial web content change, and develop scalable and effective content change prediction models.

3 ANALYZING PRICE CHANGE AT SCALE

Before diving into the details of building machine learning models for predictive crawling, we first provide a large-scale analysis of price change patterns for online offers. In particular, we focus on temporal, geographical and content patterns.

3.1 The Salticus Dataset

In order to conduct an unbiased analysis that avoids feedback loops [29], we sampled one million online offers and scheduled hourly crawls for all of them. The sample is a stratified mix of two sample types: (a) random uniform from the entire corpus of offers; (b) click weighted, to better represent popular offers that receive clicks. We download hourly snapshots of these offer pages from May 1st 2018 to the middle of September 2018, and use this crawl data for the following analysis, as well as for our predictive machine learning models. We refer to these snapshots as the *Salticus*[1] *dataset* in the remainder of this paper.

3.2 Temporal Patterns

We analyze two types of temporal patterns. First, we examine whether there are certain preferred time periods (e.g., weekday or weekend, morning or evening) for merchants to change prices. Second, we analyze how long it takes for an offer to change its price. Other temporal patterns such as changes over different days of the month and months of the year can also be interesting. However, since we only have data for a few months, such analysis might not be representative statistically.

Figure 2 illustrates the price change probability over the different days of the week and hours of the day (in merchant local time).

[1]Named after a spider genus, popularly known as "zebra spider".

We find that merchants are more likely to change prices at midnight. The peak of midnight change is probably due to automated updates. For days of the week, changes are more likely to happen on weekdays than on weekends. The non-uniform temporal patterns suggest that time is a useful factor to determine price change. Therefore, both day of the week and hour of the day are applied in our machine learning models. It is worth noting that our data spans from May to September which does not overlap with major holiday seasons. We expect different temporal patterns during holidays.

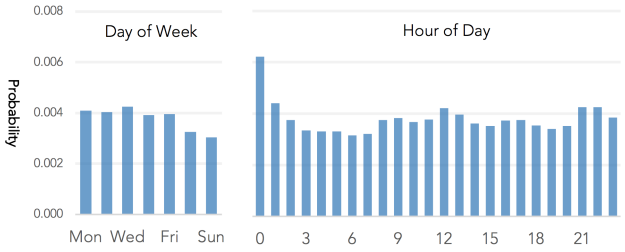

Figure 2: Probability of price change for different days of the week and hours of the day (local time). For each horizontal axis unit (e.g., Mon.), its probability equals to the number of downloaded offer pages with price change (on Mon.) over the total number of downloaded offer pages (on Mon.).

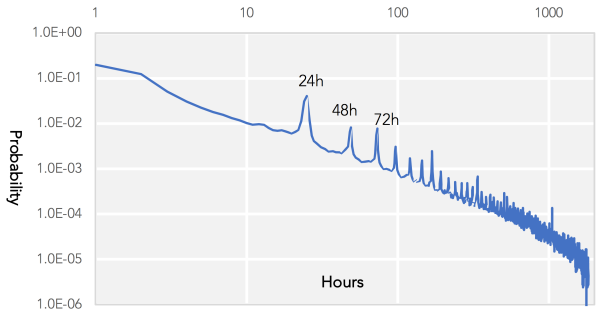

Figure 3: Distribution of time intervals between two price changes (log-log plot). The probability for a given time interval measures how many of the price changes in the Salticus dataset happened in the given interval.

Figure 3 plots the distribution of time intervals between price changes. The horizontal axis denotes the time interval (in hours) for an offer to change its price; the vertical axis denotes the percentage of price changes — the amount of price changes within a given time interval over the total number of price changes in our dataset. As an example, suppose our dataset consists of two offers, A and B. Offer A changes its price at $t_0 + 1h$ and changes again at $t_0 + 3h$. Offer B changes its price at $t_0 + 1h$. Therefore, the total number of price changes is three, with two coming from A and one from B. The probability of change for $1h$ time interval is $2/3$, and the probability of change for $2h$ interval is $1/3$. Note that for an offer, the first observation of its price does not count as a change since we do not know its prior price.

Figure 3 conveys several messages. We find that among all price changes, a considerable amount of them happen within a short time. This might come from the products whose prices depend on external factors such as price fluctuations of financial instruments. One illustrative example is gold coin prices, which change dynamically together with the gold spot price. Second, we see multiple local peaks at 24 hours, 48 hours, etc. This may be due to the automatic price updates, which was also discussed in Figure 2.

3.3 Geographical Patterns

The Salticus dataset contains online product offers from across the world. This enables us to explore price change patterns for different countries. Figure 4 provides the change probability over 20 countries. Each country is represented by its ISO code.[2] The probability is computed in the same manner as described in Section 3.2. We find that different countries exhibit varying change patterns. Among the 20 countries we considered, India (IN), Mexico (MX) and Brazil (BR) are the countries with the highest price change rate whereas the United States (US), Japan (JP) and Indonesia (ID) are the lowest. This might be related to economic factors such as inflation rates in a particular country. We find a 0.4629 Pearson correlation between countries' price change rates and their inflation rates (as of October 2018). The variation of price changes across countries motivates including country as a feature in our machine learning model.

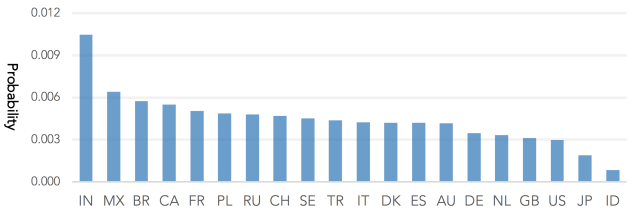

Figure 4: Price change probability in different countries. Probability is computed similarly to Figure 2.

3.4 Content Patterns

This section further explores the price change patterns for different types of products. In our repository, each offer is associated with a product category.[3] A product category such as *Apparel & Accessories > Clothing* defines the product area and the subcategory of an offer. With such information, we then analyze the price change probability in each top-level category of the taxonomy. Again, the probability is computed similarly to Section 3.2.

Figure 5 presents the price change probabilities for different taxonomy categories. *Software* has the highest change rate, as it is affected by a highly dynamic *Virtual Currency* subcategory. *Electronics* (e.g., phones, tablets) and *Media* (e.g., books, music) are more dynamic than others. Offers in *Business & Industrial* and *Religious & Ceremonial* categories have the lowest change rates. Again, the non-uniform price change rate across the different categories implies that offer metadata can provide meaningful information when building machine learning models, which we take into account when designing our model as described in the next section.

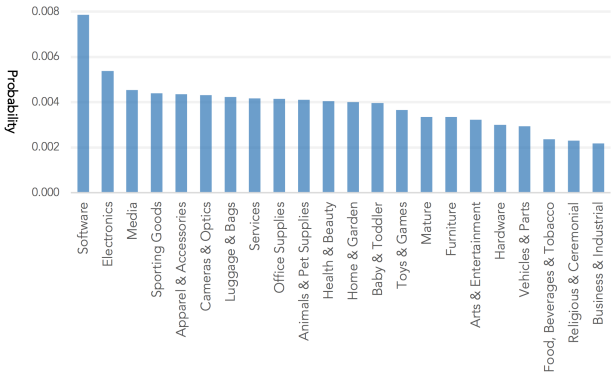

Figure 5: Price change probability over product categories. Probability is computed similarly to Figure 2.

4 METHODOLOGY

We model the price change prediction as a classification problem, for which we want to predict, for any individual offer, whether its price will change in the next K hours. Here, we do not consider the degree of change since our ultimate goal is to build an effective crawler that re-crawls an offer's page as long as there is a change. Therefore, we do not model it as a regression problem.

To build such a model, a set of training examples is required. Each example contains a list of features and a corresponding label. A positive label means that the offer's price changes within the next K hours; otherwise, a negative label is assigned. If a price x changes to y and then changes back to x, we still count it as a positive label because there are indeed price changes (twice) within the K hours. The Salticus dataset described in Section 3.1 is used for deriving our training, validation and testing examples.

4.1 Training/Testing Data

4.1.1 Generating Training Examples. The Salticus dataset contains two types of crawl events — the hourly scheduled crawls described in Section 3.1, and the regular crawls from the production system. Each crawl event is associated with a timestamp t, which allows us to split our data by time and simulate the moment we need to make a *<crawl, not crawl>* decision in the production system.

Specifically, for each simulated *prediction time* t, the future crawls after t provide full observability (on an hourly basis) of the price information in the next K hours. We use the offer information up to t to generate features, and the next K hours to generate ground-truth binary labels reflecting the price change (or lack thereof), thereby creating a single training example. By shifting t and repeating this process, we create a set of training examples.

In this paper, the *prediction time* t comes from both hourly crawls and production crawls. Using hourly crawls in addition to production crawls provides multiple benefits. First, it helps to generate

[2]ISO codes are defined at https://en.wikipedia.org/wiki/ISO_3166-1
[3]We adopt the public Google Shopping Product Taxonomy for our online offers. A full schema is at: http://www.google.com/basepages/producttype/taxonomy.en-US.txt.

enough training examples for rarely crawled offers in the production crawler. Second, it guarantees that we have an unbiased training data, with prediction times uniformly spread out over all 24 hours a day, and 7 days a week.

Finally, we observe non-uniform distributions for crawling time — crawls for popular offers are aggregated densely around the same time. To avoid generating too many similar examples, we restrict different ts to be at least 15 minutes away from each other.

The entire training generation process is illustrated in Figure 6. Instead of choosing the immediate follow-up crawl event as the next t, we skip two regular crawls and one hourly crawl since they are crawled less than 15 minutes from the last prediction time.

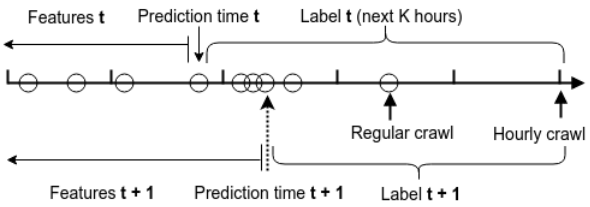

Figure 6: Training data generation process for timestamps t and $t + 1$. A vertical line denotes a Salticus hourly crawl; a circle represents a production crawl.

4.1.2 Seen and Unseen Offers. The one million Salticus offers used for model development only make up a small fraction of active offers. To understand whether our trained models can be generalized to offers that were never seen from the training data, we divide the Salticus dataset into seen and unseen offers. This is achieved by holding out 25% (i.e., 250K) of the one million offers (denoted *unseen offers*) and the remaining 750K offers are used as *seen offers*. Combining with the time-based split of training and testing data, the evaluation setup is illustrated in Figure 7. Our predictive models are only trained with the 750K offers.

Here, we use the unseen offers to simulate the production setting where our models do not observe any historical information for most of the offers. Therefore, we expect better model performances for the seen offers comparing to the unseen offers.

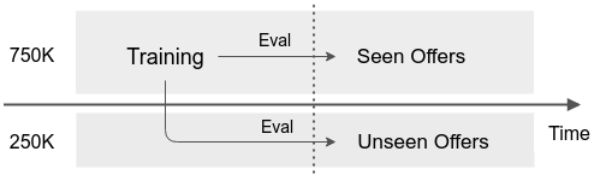

Figure 7: Overview of the training/evaluation setup.

4.1.3 Training/testing Setup. Our training/validation/testing datasets are split based on time. Particularly, crawling events from May 2018 to July 2018 are used for training, data from August 1 to August 5 is used for validation, and data from the rest of August is for testing. In total, we extract around 1.3 billion training examples, 84 million validation examples and 620 million testing examples. As expected, most of them did not contain price changes (i.e., negative price change label). In this paper, we set $K = 6$ hours, for which we observe around $1 : 48$ positive/negative ratio for all training, validation and testing datasets. The validation dataset is used for model selection, and the reported results are based on the chosen model's performance on the testing data.

4.2 Features

We consider two types of features in this paper. First, we employ the historical change frequency features that are commonly adopted in previous studies [9, 28]. Second, we propose using metadata and other offer-related information that may be predictive of a price change even for offers with limited or no prior history.

4.2.1 Offer Change Frequency. We first extract a set of price change frequency features pertaining to each individual offer. We define the change frequency as the number of price changes per hour. Depending on the amount of crawling history being used, we extract the following three types of change frequency features — change frequency within the most recent day, week and month. We extract the time since the last price change and use it as a feature, as well.

4.2.2 Product Category Change Frequency. Data sparsity is a critical issue for offer-level change frequency. For new offers and rarely-crawled offers, the change frequency is either unavailable or unreliable. Radinsky and Bennett [28], in the context of web search, proposed to learn from similar web pages when encountering the sparsity issue. Similarly, in the context of commercial content, two offers may share similar change patterns if they come from the same product category. As a result, we further compute the change frequencies at the product category level and use them as features.

4.2.3 Metadata Features. The change frequency features are highly dependent on crawl history. This introduces issues when the history information is unavailable or is subject to change. To abate these issues, we introduce a number of history-independent features in our model. As shown in Section 3, offers from different product categories and different countries tend to exhibit different change patterns. This motivates us to extract the metadata information of an offer and use it as a source of features. Besides the product category and country, we include brand, condition, language, merchant and web page title of the offer. Details of the metadata features are presented in Table 1. Our previous analysis in Figure 2 illustrates diverse price change patterns for different days of a week and hours of a day. Thus, the current day and current hour (of the prediction time) are also included as predictive features.

Compared to the offer and product change frequency features, metadata-based features have several benefits. First, they alleviate the cold-start issue since metadata is readily available for both existing and brand-new offers and product categories. Second, they are robust with respect to unexpected changes in the crawl history. Third, as we show next, metadata-based features can be combined with change frequency features for further model improvement.

4.2.4 Full Observability VS. Partial Observability. As mentioned, the change frequency features are highly sensitive to the amount of available crawl history. In the best case, if we can afford constant (e.g., hourly as in our dataset) crawls, the estimates for change frequency features will be more accurate. However, production

Table 1: Features used in baselines and the proposed models.

Price change frequency for offer O	
Frequency (1 day)	Change frequency within last day
Frequency (1 week)	Change frequency within last week
Frequency (1 month)	Change frequency within last month
Most recent change	Time since the most recent change
Price change frequency for offers in same product category C	
Frequency (1 day)	Change frequency within last day
Frequency (1 week)	Change frequency within last week
Frequency (1 month)	Change frequency within last month
Most recent change	Avg. time since the most recent change
Metadata and related information for offer O	
Brand	Brand id
Condition	Condition: new, used or refurbished
Country	Country code
Day of Week	Day of week for the prediction time
Hour of Day	Hour of day for the prediction time
Language	Offer page language
Merchant	Merchant id
Offer title	Offer page title
Product category	Product category

crawlers usually cannot achieve the full observability, both due to the crawling costs as well as the politeness constraints [23].

In this paper, we experiment with two types of observability — **full observability** where we assume the observation of hourly crawls, and **partial observability** in which we only observe crawls from the existing crawling strategy. The former tells us the upper bound, whereas the latter can help us to estimate the predictive power for the change frequency features under existing crawling strategy. It is worth noting that the partial observability is sensitive to the change of crawling strategy, and adjustments to the strategy will further affect the observability, while the full observability scenario is not practical at the scale of the entire corpus.

4.3 Models

4.3.1 Baselines. In this paper, we adopt a set of baselines purely based on the change frequency features, in which we assume that the probability an offer changes its price in the next K hours equals to the historic price change frequency. Depending on the level of granularity, we include two baselines: an offer-level price change baseline and a product category level price change baseline. The former predicts an offer's price change based on its own history, whereas the latter employs the change history of its product category. Moreover, we include a third baseline combining both offer-level and category-level features in a linear model. These simple but effective baselines were commonly adopted in previous studies [8, 9, 28, 30].

4.3.2 Proposed Models. As discussed, we model the prediction task as a binary classification problem to classify whether an offer's price will change in the next K hours. To incorporate both numerical change frequency features and metadata information in one unified framework, in this paper, we adopt a feed-forward deep neural network (DNN) model.

The change frequency features, if included, are treated as numerical dense features. Metadata information is modeled by sparse features, which are embedded into a low-dimensional space [22]. Here, an embedding is a condensed vector (low-dimensional vector) representation of metadata information. Semantically, similar metadata would have similar embeddings located closely in the vector space. The combination of dense frequency features and embedded metadata features in a single neural network architecture enables both *memorization* and *generalization*, a desired property for a predictive machine learning model [7].

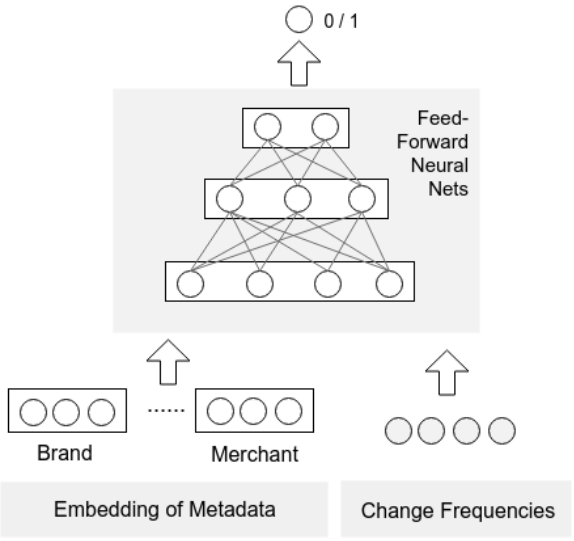

Figure 8: An illustration of our model architecture.

We use the TensorFlow DNNClassifier for model implementation [1], where we set three hidden layers with 256, 128 and 64 hidden units in each layer. We adopt ReLU (Rectified Linear Unit) as the activation function for hidden units, and choose the Adagrad algorithm to optimize the cross-entropy loss. To deal with over-fitting, we adopt both L1 and L2 regularization and set both to 0.001.

To summarize, this paper includes the following baselines (the first three) and proposes two models for predicting price changes.

- Frequency (offer): offer-level change frequency
- Frequency (category): category-level change frequency
- Frequency (combined): combines offer-level and category-level change frequencies in a linear model
- DNN (metadata): DNN using only metadata information
- DNN (metadata + frequency): DNN using both metadata and offer + product category change frequencies.

5 PREDICTING PRICE CHANGE

This section reports the comparison results between the proposed models and baselines. Our evaluation adopts the **AUC** metric (Area Under Receiver Operating Characteristic Curve) to measure the

performance of different models and baselines. A value of 0.5 means a random guess whereas 1.0 corresponds to a perfect prediction.

5.1 Evaluation on Seen Offers

Table 2 reports the testing AUCs for different predictive models on seen offers (see Section 4.1.2 for its definition). Here, we try to assess model quality on predicting the future for the offers being seen from the training set. The AUC values are obtained in the following way — firstly, we evenly split our testing data into 20 subsets; then, we evaluate on each subset; finally, we compute the averages and standard deviations over the 20 subsets. In Table 2, the top nine rows refer to baselines and the last two correspond to the proposed models. Overall, all of these models achieved AUCs above 0.5, meaning that both change frequency and offer metadata information have a predictive power. They can also be combined to further boost the prediction performance.

Table 2: Testing AUC (standard deviation) for different prediction models on seen offers.

Model \ Observability	Partial	Full
Frequency (category, 1 day)	0.560 (0.003)	0.573 (0.004)
Frequency (category, 1 week)	0.577 (0.004)	0.584 (0.004)
Frequency (category, 1 month)	0.570 (0.003)	0.583 (0.003)
Most recent (category)	0.569 (0.004)	0.582 (0.003)
Frequency (offer, 1 day)	0.642 (0.002)	0.721 (0.003)
Frequency (offer, 1 week)	0.769 (0.003)	0.815 (0.003)
Frequency (offer, 1 month)	0.791 (0.003)	0.832 (0.003)
Most recent (offer)	0.789 (0.003)	0.827 (0.003)
Frequency (combined)	0.791 (0.004)	0.847 (0.003)
DNN (metadata)	0.842 (0.005)	
DNN (metadata + frequency)	**0.875 (0.002)**	**0.883 (0.003)**

5.1.1 Full Observability VS. Partial Observability. Table 2 shows that the change frequency features computed from fully-observable crawl history outperform the ones derived from partially-observable history (details about observability are in Section 4.2.4). In terms of the baseline *Frequency (offer, 1 day)*, having fully observable history boosts AUC by as much as 12.3%. This implies the importance of collecting more crawling history. However, such an implication goes against our ultimate goal — building predictive models to lower crawling frequency. We can resonably expect that after reducing crawls from production crawlers, the performance of the partial observability baselines will decrease.

Compared to the baselines, our proposed models have the following advantages. First, the *DNN (metadata)* model does not use any price change feature and thus is not subject to the change of crawl history observability. Second, for the *DNN (metadata + frequency)* model, its performance difference under full and partial observability is small compared to the difference in the baselines. Both of them demonstrate the robustness of our proposed models.

5.1.2 Baselines VS. Proposed Models. As for baselines, we observe that offer-level price change features significantly outperform the category-level features. This is expected since the product category only provides a high-level abstraction for an offer whereas specific characteristics for each offer are missing. However, category features would be helpful when offer-level features are absent, e.g., for new offers without history. In this experiment, we only see a marginal improvement when combining offer and category features. This is because there are no new offers in the seen offers.

Comparing to the baselines, our *DNN (metadata)* model significantly outperforms the best baseline using the partial observability history, and works equivalently well as the best baseline using the full observability history. Since *DNN (metadata)* does not use any change frequency feature, it is robust towards new offers. Moreover, once the change frequency features are available, they can be simply incorporated in the *DNN (metadata + frequency)* model to yield even better performance.

5.2 Generalizability to Unseen Offers

To understand whether our models can be generalized to other offers, this section replicates the evaluation in Section 5.1 but assesses over 250K unseen offers (see its definition in Section 4.1.2). This section examines how our proposed models would generalize to the production system, which often handles billions of offers. Note that an unseen offer is different from a brand-new offer. The former refers to an offer that was not observed in the training data, whereas the latter denotes an offer without crawl history. For the unseen offers in this paper, we can still compute their offer-level and category-level price change frequency features.

Table 3 reports the AUCs of different predictive models for unseen offers. The baselines are roughly the same as Table 2, whereas the proposed models drop slightly. This aligns with our expectation since some metadata information from the unseen offers might never be observed from the training data. Surprisingly, even with partially observable history, the proposed models still outperform all baselines and maintain the AUCs above 0.80. With fully observable history, although the pure metadata feature does not achieve the AUC as the best baseline, it can be combined with the price change features and eventually produces the best performance.

It is worth noting that although we do not specifically examine our model performances over the brand-new offers, we still expect that we can achieve reasonable performances using *DNN (metatda)* since this model does not use any crawl history information. In this case, offer level baselines would fail and the *DNN (metadata + frequency)* model will be downgraded to *DNN (metadata)*.

5.3 Summary

Overall, the evaluation results from the above two sections clearly demonstrate the effectiveness of our proposed models, which not only work well on the seen offers but also show generalizability to unseen offers. Besides, we discover that metadata information is an important predictive feature. Such a feature is relatively static and easily accessible across different types of offers, effective in the cold start setting, and is not subject to the change of crawl history availability. Our proposed DNN approach provides an effective way to utilize the metadata information, and further enables the incorporation of additional features.

Table 3: Testing AUC (standard deviation) for different prediction models on unseen offers.

Model \ Observability	Partial	Full
Frequency (category, 1 day)	0.554 (0.002)	0.567 (0.002)
Frequency (category, 1 week)	0.572 (0.002)	0.579 (0.002)
Frequency (category, 1 month)	0.565 (0.002)	0.579 (0.002)
Most recent (category)	0.564 (0.002)	0.578 (0.002)
Frequency (offer, 1 day)	0.642 (0.002)	0.720 (0.003)
Frequency (offer, 1 week)	0.766 (0.003)	0.817 (0.003)
Frequency (offer, 1 month)	0.788 (0.003)	0.833 (0.003)
Most recent (offer)	0.787 (0.003)	0.828 (0.003)
Frequency (combined)	0.797 (0.003)	0.849 (0.003)
DNN (metadata)	0.803 (0.003)	
DNN (metadata + frequency)	**0.854 (0.003)**	**0.862 (0.003)**

6 PREDICTIVE RESOURCE ALLOCATION

Experiments in Section 5 demonstrate the feasibility of building machine learning models to predict price change. However, it remains unclear how such models can be integrated with production crawlers. This section attempts to answer the question.

As mentioned in the Introduction, there are two important components in our production crawler — the *Batch Scheduler* that downloads offer pages periodically, regardless of whether or not the offers are accessed by users, and the *InstaCrawl* that follows past user clicks and downloads offer pages based on their click frequencies. The design of *InstaCrawl* relies on the observation that the past click is a good predictor of the future click [14]. Indeed, in our production crawler, 95% of user-clicked offers were also accessed in the past week. Besides, the *InstaCrawl* is also in line with previous studies which account for user experience when developing user-centric crawling quality metrics [26, 32].

In this paper, we focus on the optimization for *InstaCrawl* because the corresponding offers are more likely to be accessed by users. However, we believe that our model can also be applied to the *Batch Scheduler*, which is one of our future work directions. To be specific, our predictive models will be used for reducing unnecessary *InstaCrawl* crawls that do not yield new updates to the price information. This is achieved in the following way. For each crawl request, we skip it if our model predicts that the price will not change; otherwise, we send the request for crawling. Since our current production crawler harvests all requests and the proposed models only target to crawl a subset of them, we can use existing production logs for simulation.

It is worth noting that there is a trade-off between spending resource and improving the price information freshness. If there are unlimited resources (for both crawlers and merchant servers), one can always reach the optimal freshness. When crawling resources are limited, more stale prices will be shown. However, some approaches will be able to make better trade-offs than others. In this section, we compare the trade-offs of our proposed model with the commonly-adopted baselines in the literature.

6.1 Experimental Setup

6.1.1 Overview. As discussed, with production crawler logs, we can simulate different strategies to utilize the predictive models. To understand the effectiveness of each strategy, we need a test collection that contains all of the regular production crawls for simulation, as well as the true prices for evaluation. The Salticus dataset (see Section 3.1) is such a repository.

The simulation process is illustrated in Figure 9. Each circle denotes an *InstaCrawl* request c_i from our production crawler, and each vertical line corresponds to a Salticus hourly crawl s_k. When developing a predictive crawling strategy, we apply the decision (to crawl or not) based on a machine learning prediction for each c_i. If the decision is to crawl, the price ($\text{price}_{\text{local}}$) will be updated; otherwise, it stays unchanged. The hourly crawl s_k is used to compute the ground-truth price ($\text{price}_{\text{true}}$).

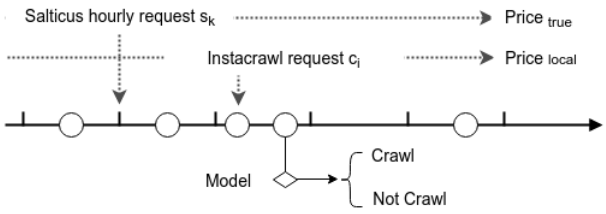

Figure 9: An illustration of applying machine learning predictions in a production crawler.

6.1.2 Crawl Decisions. Converting a model prediction to a crawl decision is a critical component in Figure 9. The simplest way is to define a threshold. A request with prediction score above the threshold will be crawled; otherwise, it will be skipped. However, a hard decision may result in some offers never being crawled. Therefore, we adopt a probabilistic decision strategy — the crawl decision is made stochastically and proportionally to the prediction score. In this way, offers with low prediction scores but high click rate still have a chance to be crawled.

The skewedness of prediction scores makes it difficult to directly use the original scores. As a result, we rank scores and map them into n quantiles (with the first quantile mapping to the lowest score). For the i-th quantile, we use the probability in Equation 1 for making the stochastic decision. This is a simple heuristic that makes the probability linearly proportional to its quantile — a larger quantile will result in a higher probability. We do believe that there are more principled ways to determine the probability; however, we only seek for the simplest way for the proof of concept in this paper. Here, γ is a resource factor, which can be tuned to achieve different resource saving percentages.

$$p_i = \min\left(1.0, \frac{i}{n} \cdot \gamma\right), \quad \gamma \geqslant 0 \tag{1}$$

6.1.3 Compared Models. We compare the following crawling strategies. First, we include a baseline that makes uniform-probability (p_u) decisions for all *InstaCrawl* requests. Different p_u can yield different resource consumption rates. For example, setting $p_u = 0.8$ will crawl any request with the probability of 0.8, and thus can save

20% of resources. The uniform baseline provides us with a basic understanding of the task difficulty.

Based on the evaluation results from Section 5, we further include the following three models: the best baseline *Frequency (combined)*, the best model *DNN (metadata + frequency)* and *DNN (metadata)*. *DNN (metadata)* does not use crawl history and thus can handle cold-start and the change of crawl history observability. The change frequency features are sensitive to crawl history observability, different observability levels will yield different feature values (see Section 5.1.1). In this section, we use partially observable history, i.e., downloaded pages from *InstaCrawl*, but not the hourly Salticus crawls. Note that when a new crawling strategy is applied, observability will change further since earlier no-crawl decisions will affect the availability of crawl history. Despite that, *DNN (metadata + frequency)* still gives us the upper bound performance.

6.1.4 Evaluation Metrics. Our project operates on the trade-off between resource and performance. When crawling resources are reduced within the same strategy, the performance will always drop. However, some models might be able to make better trade-offs than others. Therefore, comparing the trade-offs among different strategies is the primary focus for evaluation. Of course, with saved resources, one can schedule more crawls for unpopular pages, which may further improve the overall performance. However, we mainly focus on evaluating *resource savings* in this paper, and leave the *resource re-allocation* evaluation for future work.

In our evaluation, resource usage (denoted by R) within a time period $[t_s, t_e]$ is defined to be proportional to the number of crawling requests that are eventually sent for page download at this time period. To be specific, suppose that we have N *InstaCrawl* requests and C ($C \leq N$) of them are finally sent for crawling. R will be proportional to C. Since N is a constant during our evaluation of different crawling strategies, we use it for normalization, i.e., $R = C/N$. Note that reducing crawls will not only reduce the load of our production crawler but also significantly reduce the load for the merchant servers. Too many requests at the same time might even cause our production crawlers to be blocked.

For performance, we adopt the commonly-used freshness metric (denoted by F) [9, 28]. A page is considered to be fresh if its latest local copy of crawl result aligns with its real-world content. Since the *InstaCrawl* requests are correlated with the number of user clicks, we can treat each c_i in Figure 9 as a user click. Then, we compute the freshness for each click c_i **before** we actually crawl it. This way, we actually examine whether a user will see the right content after click. As shown in Equation 3, the overall freshness F is computed by aggregating the freshness of all clicks in $[t_s, t_e]$.

$$f(c_i) = \begin{cases} 1, & \text{if price}_{\text{local}} = \text{price}_{\text{true}} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$F(t_s, t_e) = \frac{1}{N} \sum\nolimits_{t_{c_i} \in [t_s, t_e]} f(c_i) \tag{3}$$

6.2 Resource Savings

In this section, we first describe a resource-freshness trade-off for the uniform baseline, aiming to provide a basic understanding of the problem space. Then, we show how the trade-off can be improved using different machine learning based crawling strategies.

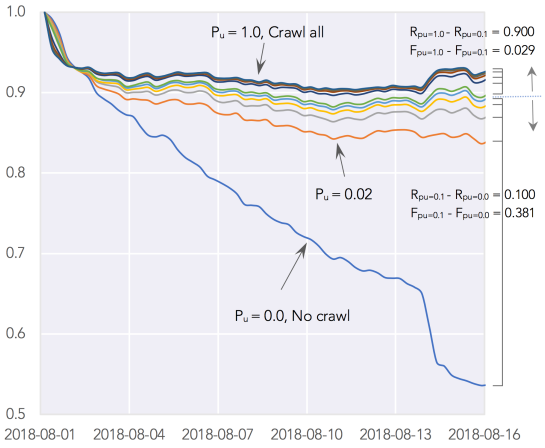

Figure 10: The change of freshness over time for a set of uniform crawling strategies. Each line denotes a p_u, which corresponds to 0.0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.3, 0.5, 0.7, 0.9 and 1.0, respectively (bottom to top).

6.2.1 Freshness-Resource Trade-offs. Unlike the evaluation in Section 5, this section attempts to assess the accumulated effects of a number of machine learning decisions over time. However, first we will forego machine learning, and try to better understand the trade-off between freshness and resource utilization in our application. On one side of the spectrum, will the freshness be 100% if we harvest every regular crawl request in Figure 9? On the other side of the spectrum, what level of freshness can be maintained if we always decide *not to crawl*. Between these extremes, we can control the freshness-resource trade-off based on the uniform probability p_u (i.e., the uniform baseline in Section 6.1.3).

Accordingly, we continuously compute the freshness for different uniform crawling strategies every six hours (i.e., t_e - t_s = 6h) starting from August 1st, and plot them in Figure 10. Here, our experiments assume everything is synchronized at the beginning, so each line starts from freshness 1.0. In Figure 10, $p_u = 0.0$ (the bottom line) denotes that we never crawl, and thus the resource consumption is 0%. $p_u = 1.0$ (the topmost line) means that we crawl at every click c_i and the resource consumption is 100%.

We observe that the first 10% of resource (p_u from 0.0 to 0.1) improves freshness significantly by 0.381, whereas a further resource investment only provides a marginal increase — an additional 90% of resources (p_u from 0.1 to 1.0) only lifts the freshness by 0.029. This clearly shows that uniform crawling heavily overspends resources, particularly after $p_u > 0.1$. To reach the same level of crawl quality, other strategies might require much less resource. Therefore, the major focus of the following section is to explore the ways of optimizing resource utilization using our proposed models.

6.2.2 Resource Saving at Different Freshness Levels. To align with Section 5, we also evaluate on both seen and unseen offers. For each offer type, we compare resource utilization at fixed freshness levels for different crawling strategies. Instead of reporting values from one specific time period, we aggregate and report evaluation metrics over ten time periods five days each, i.e., August 5 to August 9, August 10 to August 14, and so on. Here, we start from

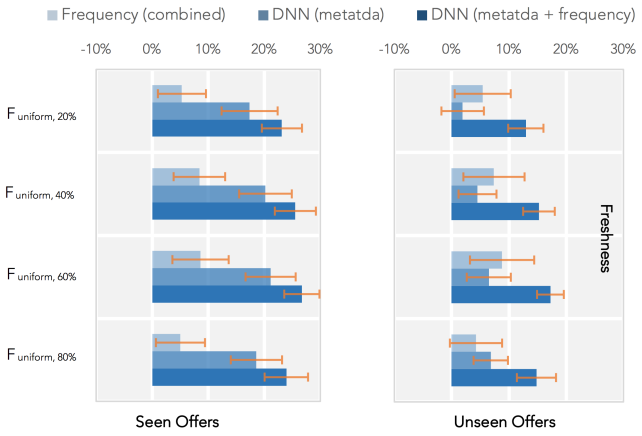

Figure 11: Resource savings (with 95% confidence interval) for different crawl strategies at target freshness levels. The horizontal axis denotes % of resources saved compared to the uniform strategy; the vertical axis denotes freshness. Freshness levels ($F_{uniform, X\%}$) are selected to cover 20%, 40%, 60% and 80% resource for the uniform strategy.

August 5 because the impact from initialization fades away (we assume a freshness of 1.0 in the beginning of our simulation) and the performance becomes more stable.

In our evaluation, the freshness levels are chosen based on the uniform strategy, where we cover the freshness corresponding to the resource usage of 20%, 40%, 60% and 80% for the uniform strategy. We name them as $F_{uniform, 20\%}$, $F_{uniform, 40\%}$, $F_{uniform, 60\%}$ and $F_{uniform, 80\%}$. For each predictive crawling strategy, the amount of resources needed for a freshness level is obtained through tuning γ in Equation 1. Then, for the given crawling strategy, we measure its *resource saving* using the relative percentage of resources that can be saved compared to the uniform strategy.

Figure 11 plots the resource savings for each crawling strategy. Here, 0% means that a crawling strategy utilizes the same resource as the uniform strategy. Compared to the uniform baseline, all other models show the ability of spending less resources while maintaining the same freshness. In particular, *DNN (metadata + frequency)* achieves the best performance, which saves as much as 27% resources for seen offers and 17% for unseen offers. *DNN (metadata)* also outperforms the *Frequency (combined)* baseline on seen offers and performs equivalently on unseen offers. This aligns with our findings in Section 5.2, demonstrating the effectiveness of our proposed machine learning models. Note that there remains a big discrepancy between the seen and unseen offers for our proposed models. This is consistent with our findings in Section 5.2 and is due to the mismatch of metadata in these two different types of offers. In the future, we will explore more effective ways to bridge this gap between seen and unseen offers.

Again, as discussed in Section 5.2, since *DNN (metadata)* does not use any crawl history information, we expect that the resource saving performance for the unseen offers can be generalized to cold-start offers with no or very little history information. Overall, in this case, *DNN (metadata)* model can still save around 5% of resources whereas all frequency baselines will fail.

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this paper, we study predictive resource allocation in the context of building a production crawler for commercial content. In particular, we focus on the problem of price change prediction for commercial offers. We start our study in Section 3 via a detailed analysis of price changes on a global scale. To the best of our knowledge, this is the first such publicly available analysis. In addition, the predictability of price change dynamics that we discover, motivates the further development of predictive price change models.

In Section 4, we propose a deep neural network based approach for predicting future price changes. The model goes beyond the change frequency features commonly used in prior work, by incorporating metadata information of online offers. There are multiple contributions of this approach. First, to the best of our knowledge, this is the first published attempt to build predictive crawlers in the context of commercial web content and price change prediction. Second, we update the existing predictive models [28, 30] with a state-of-the-art machine learning approach, which incorporates both numerical frequency features and content-based sparse features within a unified framework. Experimental results in Section 5 demonstrate the effectiveness of our approach. Moreover, the incorporation of metadata enables our model to deal with the cold-start problem and avoid feedback loops.

Beyond simply predicting price changes, we further study models integration with a production crawler in Section 6. Evaluation with production crawl logs demonstrate that our models can make better resource-freshness trade-off decisions than the frequency-only baselines both for seen and unseen offers. Furthermore, through the use of metadata information, our models can save resources even for offers with no prior price change history.

7.2 Future work

Despite the promising results we obtained, our study still has several limitations, which will be the focus of our future work.

First, we observe that models built with solely metadata information perform less well on unseen offers. This can be partially solved by increasing the size of the full-observability Salticus dataset. However, for truly new offers, an exploration of more advanced approaches for a better modeling of metadata information will be required. Besides, we see evidence that more frequent model retraining increases the wins and plan to adopt it in the future.

Second, our use of machine learning scores for predictive resource allocation may be sub-optimal. Our experiment adopts a simple quantile-based conversion from prediction score to sampling probability (see Equation 1). This can be improved by adopting a more direct optimization approach. In addition, we may also explore ways to productively re-allocate the crawl resources saved by our methods in future work.

Finally, in this paper, we made the simplifying assumption that price changes (and crawls) can be modeled independently, and do not utilize any sequence information. This allows for simpler model deployment, and, as demonstrated in Sections 5 and 6, already outperfoms the existing baselines. Thus, an interesting direction for future work is applying sequence based deep learning models (e.g., RNN, LSTM [17]), to further improve the model performance.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Vol. 16. 265–283.

[2] Eytan Adar, Jaime Teevan, Susan T Dumais, and Jonathan L Elsas. 2009. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*. ACM, 282–291.

[3] Yossi Azar, Eric Horvitz, Eyal Lubetzky, Yuval Peres, and Dafna Shahaf. 2018. Tractable near-optimal policies for crawling. *Proceedings of the National Academy of Sciences* 115, 32 (2018), 8099–8103.

[4] Brian E Brewington and George Cybenko. 2000. How dynamic is the web? 1. *Computer Networks* 33, 1-6 (2000), 257–276.

[5] Maria Carla Calzarossa and Daniele Tessera. 2015. Modeling and predicting temporal patterns of web content changes. *Journal of Network and Computer Applications* 56 (2015), 115–123.

[6] Carlos Castillo. 2005. Effective web crawling. In *Acm SIGIR Forum*, Vol. 39. ACM, 55–56.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[8] Junghoo Cho and Hector Garcia-Molina. 2000. The Evolution of the Web and Implications for an Incremental Crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00)*. Morgan Kaufmann Publishers Inc., 200–209.

[9] Junghoo Cho and Hector Garcia-Molina. 2000. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD '00)*, Vol. 29. ACM, 117–128.

[10] Junghoo Cho and Hector Garcia-Molina. 2003. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)* 28, 4 (2003), 390–426.

[11] Junghoo Cho and Hector Garcia-Molina. 2003. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)* 3, 3 (2003), 256–290.

[12] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems* 30, 1-7 (1998), 161–172.

[13] Junghoo Cho and Alexandros Ntoulas. 2002. Effective change detection using sampling. In *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB Endowment, 514–525.

[14] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. Click models for web search. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 7, 3 (2015), 1–115.

[15] Edward G Coffman Jr, Zhen Liu, and Richard R Weber. 1998. Optimal robot scheduling for web search engines. *Journal of scheduling* 1, 1 (1998), 15–29.

[16] Edith Cohen and Haim Kaplan. 2001. Refreshment policies for web content caches. In *IEEE INFOCOM 2001 - The Conference on Computer Communications - Twentieth Annual Joint Conference of the IEEE Computer and communications Societies*, Vol. 3. IEEE, 1398–1406.

[17] Li Deng. 2014. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing* 3 (2014).

[18] Jonathan Eckstein, Avigdor Gal, and Sarit Reiner. 2008. Monitoring an information source under a politeness constraint. *INFORMS Journal on Computing* 20, 1 (2008), 3–20.

[19] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. 2003. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International Conference on World Wide Web*. ACM, 669–678.

[20] Carrie Grimes, Daniel Ford, and Eric Tassone. 2008. Keeping a Search Engine Index Fresh: Risk and optimality in estimating refresh rates for web pages. *Proc. INTERFACE* (2008).

[21] Damien Lefortier, Liudmila Ostroumova, Egor Samosvat, and Pavel Serdyukov. 2013. Timely crawling of high-quality ephemeral new content. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. ACM, 745–750.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[23] Marc Najork and Allan Heydon. 2002. High-performance web crawling. In *Handbook of Massive Data Sets*. Springer, 25–45.

[24] Christopher Olston and Marc Najork. 2010. Web crawling. *Foundations and Trends® in Information Retrieval* 4, 3 (2010), 175–246.

[25] Christopher Olston and Sandeep Pandey. 2008. Recrawl scheduling based on information longevity. In *Proceedings of the 17th International Conference on World Wide Web*. ACM, 437–446.

[26] Sandeep Pandey and Christopher Olston. 2005. User-centric web crawling. In *Proceedings of the 14th International Conference on World Wide Web*. ACM, 401–411.

[27] Sandeep Pandey, Krithi Ramamritham, and Soumen Chakrabarti. 2003. Monitoring the dynamic web to respond to continuous queries. In *Proceedings of the 12th International Conference on World Wide Web*. ACM, 659–668.

[28] Kira Radinsky and Paul N Bennett. 2013. Predicting content change on the web. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. ACM, 415–424.

[29] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*. 2503–2511.

[30] Qingzhao Tan and Prasenjit Mitra. 2010. Clustering-based incremental web crawling. *ACM Transactions on Information Systems (TOIS)* 28, 4 (2010), 17.

[31] Qingzhao Tan, Ziming Zhuang, Prasenjit Mitra, and C Lee Giles. 2007. Efficiently detecting webpage updates using samples. In *International Conference on Web Engineering*. Springer, 285–300.

[32] Joel L Wolf, Mark S Squillante, PS Yu, Jay Sethuraman, and Leyla Ozsen. 2002. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International Conference on World Wide Web*. ACM, 136–147.