Distributed context discovering for predictive modeling

Rong, Z.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Web Engineering Group

Distributed Context Discovering for Predictive Modeling

Master Thesis

Zheyi Rong

Supervisors:

dr. Mykola Pechenizkiy
dr. Jeroen De Knijf

Committee Members:

dr. Mykola Pechenizkiy
dr. Jeroen De Knijf
prof. dr. Paul De Bra
dr. Alessandro Di Bucchianico
dr. Herman Haverkort

Eindhoven, November 2013

Abstract

Click prediction has applications in various areas such as advertising, search and online sales. Usually user-intent information such as query terms and previous click history is used in click prediction.

However, this information is not always available. For example, there are no queries from users on the webpages of content publishers, such as personal blogs. The available information for click prediction in this scenario are implicitly derived from users, such as visiting time and IP address. Thus, the existing approaches utilizing user-intent information may be inapplicable in this scenario; and the click prediction problem in this scenario remains unexplored to our knowledge. In addition, the challenges in handling skewed data streams also exist in prediction, since there is often a heavy traffic on webpages and few visitors click on them.

In this thesis, we propose to use the pattern-based classification approach to tackle the click prediction problem. Attributes in webpage visits are combined by a pattern mining algorithm to enhance their power in prediction. To make the pattern-based classification handle skewed data streams, we adopt a sliding window to capture recent data, and an undersampling technique to handle the skewness.

As a side problem raised by the pattern-based approach, mining patterns from large datasets is addressed by a distributed pattern sampling algorithm proposed by us. This algorithm shows its scalability in experiments.

We validate our pattern-based approach in click prediction on a real-world dataset from a Dutch portal website. The experiments show our pattern-based approach can achieve an average AUC of 0.675 over a period of 36 days with a 5-day sized sliding window, which surpasses the baseline, a statically trained classification model without patterns by 0.002. Besides, the average weighted F-measure of our approach is 0.009 higher than the baseline.

Therefore, our proposed approach can slightly improve classification performance; yet whether this improvement worth deployment in real scenarios remains a question.

To Jing Liu and my parents.

Acknowledgement

I would like to express my sincerely gratitude to my supervisor, dr. Mykola Pechenizkiy, for his offering me the opportunity of working in the CAPA(Context Aware Predictive Analytics)[1] project, continuous support during my work, and the crucial arrangements of timing. I would like to express my greatest thanks to my daily tutor, dr. Jeroen De Knijf, for his great patience in guiding me, insightful comments on the data and report that I provided, and numerous meetins and enjoyable discussions he had with me on research problems. I also want to thank dr. Mario Boley for his innovative ideas of the post-processing algorithm used in my thesis.

I would like to thank the rest of my thesis committee: prof. dr. Paul De Bra, dr. Herman Haverkort, and dr. Alessandro Di Bucchianico, for their valuable suggestions, and for their help in assessing my thesis.

I would like to thank SURSara[2], the national HPC and e-Science support center in the Netherlands, for offering their cluster for my experiments.

I would like to express my appreciation to my parents Jianzhong Rong and Aihua He, for their priceless encouragement and comforting. Besides, I would like to extend my thanks to my friends: Mengyi Gong, Zijian Xu, Hang Liu, Luo xi, Hao Gao, Yu Yi and Le Chen. Finally, I am deeply indebted to Jing Liu, for her proofreading of my thesis.

[1] http://www.win.tue.nl/~mpechen/projects/capa/
[2] https://www.surfsara.nl/

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the proliferation of the Internet, click prediction, i.e., predicting the next click of a user, has received increasing attention. Prediction of the clicks on search engine result pages can be the reflection of a user's judgment of search results; hence results can be ranked descendingly according to the probabilities of predicted clicks. It also plays a crucial role on sponsored search, which is a multi-billion dollar business that generates most of the revenue for search engines. In sponsored search, the prediction is used to influence ranking, filtering, placement, and pricing of ads [10], by selecting ads with high probabilities of being clicked and placing them in a prominent position in descending order of these probabilities.

Click prediction has been well studied in search engine result pages and sponsored search. Previous studies usually take advantage of two major kinds of information for click prediction, i.e., *a)* relevance information representing the similarity between queries and webpages, and *b)* historical click-through information representing a users' previous preferences on ads [43]. These existing works mainly focus on interpreting ad clicks in terms of what users seek (i.e., relevance information) and how users choose to click (historically clicked-through information).

The information on queries and users has indeed shown its power in click prediction on search engine result pages and ads [10, 42], however, this information may not always be accessible. For example, on the webpages of content publishers , there are usually no queries from a user; even worse, user information such as age and gender is not presented if the user is not registered on this website. The only available information for click prediction are attributes implicitly derived from users, such as visiting times, IP addresses, and browser versions. Therefore, click prediction on such webpages may need additional effort than that on search engine result pages with explicit user information.

The classification using interrelated attributes may have a better performance than that using individual attributes, for attributes are often not independent from each other. This motivates us to group attributes effectively , and then use the discriminative groups to boost classification performance in click prediction. This idea is essentially behind the pattern-based classification approach, which utilizes frequent discriminative patterns , where a pattern is a group of attribute values. The effectiveness of this approach in classification has been exploited by previous studies in different domains, including associative classification, graph classification, text categorization and protein classification [11].

In this thesis, we take advantage of patterns to predict user clicks on index pages, by integrating discriminative patterns mined from historical datasets into classification modeling over big data streams. The effectiveness of patterns in click prediction will be experimentally justified in Chapter 5.

1.1 Problem Setting

The click prediction problem in our setting is to predict whether a user will click on any link, while the user is visiting the index page of a website.

Click prediction is usually formulated as a supervised learning problem [10], which is the task of inferring a function, i.e. classification model, from labeled data and then applying this classification model to predict new data. Historical data, usually pageviews described by attributes such as IP address and visiting time of user visits, are used to train classification models with a binary label of two categories: *click* and *non-click*; and then the trained models will classify incoming pageviews on an index page into these two categories. Usually, the historical data cannot fit into main memory; and new pageviews are continuously being generated at a high speed. Formally, assume there is a set of n training instances, $D = \{\boldsymbol{x}_i, c_i\}_{i=1}^{n}$, where $\boldsymbol{x}_i$ is an instance and c_i is the corresponding class label (1: *click* or 0: *non-click*). Given a new instance $\boldsymbol{x}$, the problem is to calculate its corresponding class label c.

The initial challenge of our click prediction problem is that the available attributes used as features are weak in classification. Moreover, pageviews coming at a high speed lead to the difficulties existing in handling big data streams. These challenges are described below.

1.1.1 Weak Features

Click prediction in sponsored search and advertisements are studied in [10, 42], with promising results in their experiments. However, their methods may not be suitable to tackle the problem in our setting, for the lack of query-related and user-specific features. A feature, in the context of classificaion, is a measurable representative property of an object. Specifically, query terms together with intent-related features (e.g. intent type: shopping and recreation) are important in predicting advertisement clicks [42]; and user-specific click feedback features and demographic features can significantly improve the classification performance of the click prediction in sponsored search [10].

Whereas, these features do not exist when an anonymous user directly opens the index page of a website. Instead, only limited information of the user is available in his/her visit, including his/her visiting time, IP address, screen resolution, and user agent which comprises the information of his/her browser and operating system. Unlike query terms, this information is not intentionally generated by users, thereby neither directly related to users' intents nor their clicks. Moreover, this information cannot represent previous behaviors of a user since user identity is not recorded. Therefore, the available features in our setting tend to be less representative than user-specific ones and query terms.

In addition to the challenge caused by weak features, other challenges in streaming big data also apply, as described in the following.

1.1.2 Streaming Big Data

Traditional challenges in mining over big data streams still also in our setting, since the data of our click prediction problem is a continuous pageview stream at a high speed. For example, the visit rate on one famous Dutch website is 46 times per second, and 4 million per day on average.

The high velocity of the data stream requires a fast predictive modeling in training, as well as in predicting coming data. Furthermore, models should be updated since there might be concept drift in streaming data.

Additionally, the pattern-based classification approach raises a sub-problem, namely discovering useful patterns over big data streams. Discriminative frequent patterns have shown their power in classification; however, mining them from large datasets remains unaddressed. Traditional pattern mining algorithms assume that datasets fit into main memory, thereby not that useful in our setting. Hence, new algorithms that can mine discriminative patterns in an out-of-memory fashion should be developed, and be integrated into our pattern-based approach..

1.1.3 Skewed Class Distribution

The class distribution of the pageview data stream is highly skewed. In reality, most users only take a glance at the index page of a website, then leave without any click, leading to this highly skewed class distribution. For instance, there are only around 4% people who clicked after visiting the index page of a Dutch portal website. One detriment of this skewed distribution is that a classification model may only take the majority class into consideration. For example, a classification model can achieve a nearly 100% accuracy if it classifies all instances as the majority class; but this model is rarely useful in reality.

There are various methods that successfully address the classification problem over skewed data streams. We integrate existing methods into our pattern-based classification approach to tackle the click prediction problem, as described in Chapter 4.

1.2 Approach and Contributions

In this thesis, we study discriminative pattern-based classification in a streaming big data setting. The challenges described above are addressed. In particular, *1)* weak features are properly grouped to increase their discriminative powers by discriminative pattern mining coupled with a proposed post-processing procedure; *2)* discriminative patterns in large datasets can be mined by our proposed distriubted pattern sampling algorithm in a distributed out-of-memory fashion using a reservoir sampling technique; *3)* pattern mining is integrated into streaming data classification using the sliding window technique; and *4)* problems caused by skewed class distribution are addressed by an undersampling technique. Finally, these methods are experimentally evaluated in a case study.

1.2.1 Contributions

The contributions of this thesis are summarized below.

1. We have parallelized a pattern sampling algorithm by utilizing reservoirs with limited memory, enabling mining discriminative patterns from large datasets.
2. We have leveraged pattern-based classification approach to a skewed streaming data setting, i.e. click prediction.

1.3 Organization of This Thesis

The remainder of this thesis is organized as follows. Chapter 2 gives the preliminary knowledge of classification, pattern mining, and pattern-based classification. Chapter 3 presents a solution of mining discriminative patterns from large datasets. Chapter 4 describes a streaming classification approach with the help of discriminative patterns. Chapter 5 employs the pattern-based approach to a real-world setting, and experimentally evaluates this approach. Chapter 6 concludes the thesis.

Chapter 2

Click Prediction as a Classification Task

The pattern-based classification approach defines discriminative patterns as features, which may boost classification performance and make predictive results more semantically understandable. This approach usually consists of two steps: pattern mining and model learning, both of which are important problems and well studied in data mining and machine learning. In this chapter, a brief review of these two problems is presented in Section 2.1 and 2.2; followed by an introductory study of the combination of the two problems — the pattern-based classification approach.

2.1 Classification

Classification is the task of assigning objects to one of several predefined categories. The input data for a classification task is a collection of labeled instances, each of which is characterized by a tuple $(\boldsymbol{x}, c)$, where $\boldsymbol{x}$ is the attribute vector and c is a special attribute, designated as the class label (also known as category or target attribute) [40]. x can be an individual instance without any label. The value types of attributes can be either continuous or discrete, while the class label must be discrete.

Formally, the classification task can be stated as: given training instances $\{(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_n, c_n)\}$, learn a *target function* $h : \mathcal{X} \to \mathcal{C}$ which maps an instance $\boldsymbol{x} \in D$ to the class label $c \in \mathcal{C}$. The target function is also known informally as a *classification model*. A classification model can serve as an explanatory tool to distinguish between objects of different classes; and it can also be used to predict the class label of unknown instances.

A classification technique (or *classifier*) is a systematic approach to building classification models from an input data set. Examples include decision tree classifiers [38], rule-based classifiers, neural networks [26], support vector machines [13], and naive Bayes classifiers. Each technique employs a *learning algorithm* to identify a model that best fits the relationship between the attribute set and class label of the input data. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of instances it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previously unknown instances.

A general approach for solving classification problems usually starts with building a classification model on a *training set* consisting of instances whose class labels are known. Then this built model is subsequently applied to the *test set*, which consists of instances with unknown labels.

2.1.1 Performance Metrics

Evaluation of the performance of a classification model is based on the counts of test instances correctly and incorrectly predicted by the model. These counts are tabulated in a table known as

a *confusion matrix*. Table 2.1 depicts the confusion matrix for a binary classification problem, in which each entry denotes the number of instances from an actual class classified to be a predicted class. For example, *true positive* is the number of correct predictions that an instance is positive. Based on the entries in the confusion matrix, the total number of correct predictions by the model is $(tp + tn)$, and the total number of incorrect predictions is $(fp + fn)$.

Table 2.1: Confusion matrix for a binary classification

		Actual	
		Positive	Negative
Predicted	Positive	true positive (tp)	false positive (fp)
	Negative	false negative (fn)	true negative (tn)

Different metrics of classification performance can be then defined, such as *accuracy*:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{tp + tn}{tp + fp + fn + tn}$$

For both the positive class and negative class, *precision* and *recall* can be defined. Precision is the fraction of retrieved instances that are relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved. Particularly, for the positive class, the precision and recall are:

$$\text{1-Precision} = \frac{tp}{fp + tp}, \quad \text{1-Recall} = \frac{tp}{tp + fn}$$

While for the negative class, the corresponding precision and recall are:

$$\text{0-Precision} = \frac{tn}{fn + tn}, \quad \text{0-Recall} = \frac{tn}{tn + fp}$$

A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional *F-measure*:

$$F = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

This is also known as the F_1 measure, because recall and precision are evenly weighted. For both the positive and the negative class, there is an F-measure respectively, denoted as 0-F and 1-F.

The *weighted average F-measure* then can be defined to indicate the overall performance:

$$\overline{F} = \text{1-}F \cdot \frac{\#\text{positives}}{\text{Total number of instances}} + \text{0-}F \cdot \frac{\#\text{negatives}}{\text{Total number of instances}} \tag{2.1}$$

The performances of classifiers depend greatly on the application constraints, assumptions about datasets, and characteristics of the data to be classified. There is no single classifier that works best, if no prior assumptions about the nature of the classification are made.

2.2 Pattern Mining

Pattern mining is a data mining method that involves finding existing patterns in data. A pattern can be an itemset, a subsequence, or a substructure in datasets. In the context of this thesis, we refer patterns to *itemsets* only. Examples of pattern mining include frequent pattern mining which identifies sets of items that appear together in a transaction dataset with a minimum occurence threshold. Frequent pattern mining was first proposed by Agrawal et al. [2] to analyze customer buying habits by finding associations between the different items that customers place in their shopping baskets. Such information can lead to increased sales by helping retailers do selective marketing and arrange their shelf space. Another example of pattern mining is emerging pattern mining [17] which discovers itemsets whose occurrence times increase significantly from one dataset

to another. Emerging patterns can capture emerging trends in timestamped databases or useful contrasts between data classes.

Pattern mining is formally defined as follows. Let I be a set of items, i.e. a set of binary elements. Moreover, let $D = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ be the transaction dataset, i.e. a bag of sets. Each record in D is a subset of I, i.e. $\boldsymbol{x}_i \in D : \boldsymbol{x}_i \subseteq I$. (We reuse symbole $\boldsymbol{x}$ since itemsets can technically be represented as an attribute vector.) Given a dataset D over I, the pattern space $\mathcal{L}(D)$ is the power set $\mathcal{P}(I)$ of the items. Intuitively, a pattern space consists of all the combinations of the items in a given set of items.

An *interestingness measure* of a pattern space $\mathcal{L}(\cdot)$ is a function

$$q : \{(D, \alpha) : D \text{ a transaction dataset}, \alpha \in \mathcal{L}(D)\} \to \mathbb{R}^+, \tag{2.2}$$

which quantifies the interestingness of a pattern α against a transaction dataset D.

However, often there is a fixed dataset that is clear from the context. In such cases, we can simplify the notation of an interestingness measure as $q(D, \cdot) = q(\cdot)$ by omitting the first argument.

Given an interestingness measure q over a dataset D and a minimum interesting threshold θ, pattern mining is the task of finding a collection P of interesting patterns:

$$P = \{\alpha \in \mathcal{L}(D) \mid q(D, \alpha) \geq \theta\}. \tag{2.3}$$

Following the work of Boley et al. [7], three types of interestingness measures are considered here: frequency, area and discriminativity.

- *Frequency* is the most well known and used measure, and is equivalent to the proportion of the support set of a pattern to the dataset, i.e. the number of records in which a pattern occurs to the number of all the records in the dataset. More formally, for a pattern α, the support set of α in D denoted as D_α is defined as the multiset of all records in D in which all elements of α are contained, i.e. $D_\alpha = \{\boldsymbol{x} \in D : \alpha \subseteq \boldsymbol{x}\}$. The interestingness measure *support* is $q_{\text{supp}}(D, \alpha) = |D_\alpha|$, and the interestingness measure frequency is $q_{\text{freq}}(D, \alpha) = \frac{D_\alpha}{D}$.

- *Area* [23] is another interestingness measure considered here: $q_{\text{area}}(D, \alpha) = |\alpha| \cdot |D_\alpha|$. Intuitively, this measure corresponds to those patterns that optimize for both the length of a pattern (i.e. contains many items from I) and its support.

- *Discriminativity* is applicable in a supervised setting: assume that for every record $\boldsymbol{x} \in D$, a binary class label, either positive or negative, is assigned, then let D^+ denote the records that belong to the positive class and D^- the records with a negative class label. The discriminativity measure favors patterns that have high support in one class and low support in the other class. This type of patterns is well studied, for example in the setting of emerging patterns [16] and contrast set mining [4]. The discriminativity for the positive class is then formally defined as: $q^+_{\text{disc}}(D, \alpha) = |D^+_\alpha| \cdot |D^- \setminus D^-_\alpha|$; and for the negative class, it is $q^-_{\text{disc}}(D, \alpha) = |D^-_\alpha| \cdot |D^+ \setminus D^+_\alpha|$.

Mining interesting patterns from a large dataset is challenging since the search space is exponential to the number of distinct single items. Moreover, determining the interestingness of a pattern against a dataset needs one complete scan of this dataset, which costs considerable time. Therefore, exhaustively enumerating all patterns and selecting the ones of the most interest in a large dataset is unlikely to be acceptable. In fact, existing pattern mining algorithms tend to search in such a large space in heuristic ways, such as the influential frequent pattern mining algorithm Apriori proposed by Agrawal et al. [3]. The Apriori algorithms utilizes a downward closure property to minimize the search space: an itemset is frequent only if all of its sub-itemsets are frequent. So that it proceeds by identifying the frequent individual items and extending them to larger and larger itemsets as long as those itemsets appear sufficiently frequent in the dataset. Han et al. [25] devised an FP-growth method that mines the complete set of frequent itemsets without candidate generation in a divide-and-conquer way using a special data structure named frequent-pattern tree (FP-tree), which retains the itemset association information.

Although there is an abundant literature dedicated to efficient frequent pattern mining algorithms, the key problem is not the efficient mining of the complete set of frequent patterns, which is usually very huge but not so useful. Instead, the key problem is the effective identification of a rather small but truly interesting set of frequent patterns, i.e., the interpretable (or comprehensible) and applicable (e.g., useful in classification) patterns, and the exploration of their applications in multiple domains. Furthermore, traditional pattern mining algorithms tend to exhaustively enumerate item combinations, and assume that datasets and employed data structures fit into main memory. In Chapter 3, we discuss these issues in more detail, and propose an algorithm that can mine interesting patterns from large datasets.

2.3 Pattern-based Classification

The application of frequent patterns in classification appeared in sporadic studies and achieved initial success in the classification of relational data, text documents and graphs. The main idea of pattern-based classification is that patterns define new features, which can be used in a classification model.

Figure 2.1 illustrates a simple example of the pattern-based classification. In the left table, each row represents an instance with binary attributes x and y, plus its class label c. A blue cell means value 1, otherwise 0. Suppose we have mined one pattern $\{x = 1, y = 1\}$. Whether this pattern is present or not in an instance can be seen as a feature of that instance. Adding this feature to the original data table, an augmented data table is obtained as shown in the right in Figure 2.1. Subsequently, the augmented dataset is linearly separable in $(x, y, \overline{xy})$ by function $f((x, y, \overline{xy})^T) = x + y - 2\overline{xy}$.

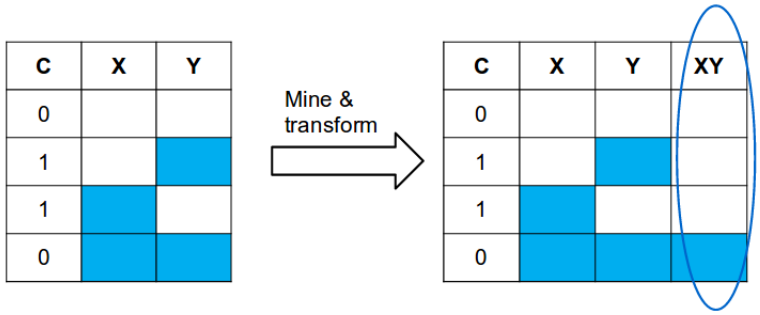


Figure 2.1: Pattern-based classification on transaction data

The pattern-based classification problem is formally defined as follows. Let D be a training dataset, $I = \{i_1, i_2, \ldots, i_m\}$ be the set of distinct items, and $C = \{c_1, c_2, \ldots, c_k\}$ be the set of class labels. Assume D contains a set of n labeled training instances $D = \{\boldsymbol{x}_i, c_i\}_{i=1}^n$, where $\boldsymbol{x}_i \subseteq I$ is a set of items and $c_i \subseteq \mathcal{C}$ is a class label. Given a set of patterns P mined according to an interestingness measure q, D is mapped into a new dataset $D' = \{\boldsymbol{x}'_i, y_i\}_{i=1}^n$, where $\boldsymbol{x}'_i \subseteq I \cup P$.

Pattern-Based Classification is learning a classification model in the feature space of single features as well as interesting patterns, where interesting patterns are generated w.r.t an interestingness measure q.

The pattern-based classification approach usually includes two steps: *1)* pattern mining, *2)* model learning. In the pattern mining step, the dataset is partitioned according to the class label, and patterns are subsequently generated in each partition with regard to an interestingness measure, such as frequency or discriminativity. The collection of mined patterns P is the feature candidates. Pattern post-processing(usually denoted as feature selection) often follows after patter mining. The set of selected features is P_s. With P_s , the dataset D is transformed to D'. The feature space includes all the individual features as well as the selected patterns. Finally, a classification model is built on the dataset D'. A general learning algorithm could be used as the

classification model. Before the classification model is applied to the test instances for prediction, the test instances will be transformed into the new feature space accordingly.

2.4 Summary

We reviewed two key problems in data mining and machine learning: classification and pattern mining. In classification, models are trained on labeled instances with known labels, to predict instances with unknown labels. In addition, different evaluation metrics of classification performance were described. In pattern mining, itemsets are mined according to user-specific interestingness measures. A combination of these two tasks is pattern-based classification, which defines patterns that can differentiate between classes as new features for classification.

Chapter 3

Distributed Pattern Mining

3.1 Introduction

The pattern-based classification approach to click prediction starts with discovering useful patterns from data. A variety of algorithms have been proposed to address this discovering as described in Section 2.2 and 3.5; however, one of the traditional assumptions that data fit into main memory no longer applies in some situations due to the enormous growth of available data size. For example, web accesses log data for a popular Internet portal in the Netherlands consists of 200GB for only two months, let alone the multitude of datasets of interest for popular social media services such as Facebook, Twitter and Youtube. This "big data" setting causes severe problems for traditional itemset mining algorithms to be applied in this setting. The main reasons for this are:

1. Most frequent itemset mining algorithms assume that the transactional database fits into main memory, which is obviously invalid in the big data setting.
2. In general, frequent itemset mining algorithms perform an exhaustive search over a huge pattern space. Although this is in principle possible for big data, it will most likely lead to severe computational cost. One popular approach to handle the high computational costs is to use parallelized frequent itemset mining algorithms.

Related to expensive enumeration of all frequent patterns, for most applications only a few patterns can effectively be utilized. Hence, the computational burden devoted on enumerating all frequent patterns is wasted on patterns that will be ignored. Moreover, deriving and storing all frequent patterns from big data may lead to severe problems, because the output size of all frequent patterns if often multiple times of the size of the input data when given a low minimum support parameter.

The work by Boley et al. [7] recently proposed a sampling algorithm that directly samples k desired patterns from the pattern space. The advantage of this approach is that the sampling algorithms achieve near optimal time complexity per pattern as well as the ability to control the distribution of the produced patterns. However, most frequent itemset mining algorithms the direct sampling approach assumes that the input data fit into main memory, which makes it inapplicable for large datasets.

A popular paradigm to efficiently extract and compute information from large datasets is the MapReduce framework. Emerged from Google's needs to efficiently compute over extreme large sets of raw Internet data, MapReduce is designed to efficiently distribute the task at hand over large distributed computing facilities. In particular, MapReduce provides the user with an abstraction layer over the physical cluster and handles inter-machine communication, partitioning the data and job scheduling [14]. In a typical setting a MapReduce task runs on a cluster of commodity machines, which can be extended by adding resources on demand.

In this chapter we extend the direct pattern sampling method proposed by Boley et al. [7], such that it is able to cope with large datasets. Specifically, we transform the method into the Hadoop

MapReduce framework, resulting in a distributed parallel frequent itemset mining algorithm with low memory demands. We performed extensive empirical evaluation on real-world and synthetic datasets to investigate the scalability and speedup of the MapReduce implementation.

In this chapter we will show that:

- The Hadoop MapReduce implementation of the frequency and area-based sampling algorithms obtain a good speedup, which makes these approaches suitable for big data.
- The memory requirements are only dependent on the size of the output data.

The rest of this chapter is organized as follows. In section 3.2 basic concepts and notation are introduced, and a brief description of the direct sampling algorithm [7] is given. In the following section we present our adaption of the sequential direct sampling algorithm into the MapReduce framework and discuss the memory usage of the method. Moreover, we also discuss implementation details and point out their affection on the performance of the system. Section 3.4 is concerned with the extensive experimental evaluation of the method. In particular we perform our method on synthetic and real datasets on a large (up to 500 mappers) Hadoop MapReduce cluster. In the final section we draw conclusions and give directions for further research.

3.2 Preliminaries

In this section we introduce the concepts and notations used in the rest of this chapter. Moreover, we will briefly describe the sequential two-step direct sampling method proposed by [7].

In addition to the notations in Section 2.2, let $\mathcal{P}(X)$ denote the power set of a finite set X and let $u(X)$ denote the uniform probability distribution on X. Moreover, for positive weights $w : X \to \mathbb{R}^+$ let $w(X)$ denote the distribution on X obtained by normalizing w, i.e., the distribution described by $x \mapsto w(x)/\sum_{x' \in X} w(x)$—assuming that there is an $x \in X$ with $w(x) > 0$.

The interestingness measures considered here are frequency, area, and discriminativiy.

3.2.1 Sequential Sampling Algorithms

With the notation and concepts previously introduced, we are now able to describe the sequential two-step direct sampling method [7]. The general outline of the algorithm is as follows: 1) first one select a data record (tuple of data records in case of discriminativity) proportional to the probability distribution of interest, 2) followed by a sampled subset of the previously selected data record (or tuple of data records) again proportional to the probability distribution of interest. A important observation of this approach is that one first have to compute the weights of all records, or tuples in case of discriminativity, and then sample the k records (tuples) from which the subsamples are reported as patterns of interest. Hence, in case of the interestingness measure of frequency is used a preprocessing that is linear w.r.t transaction database is needed, while in case of discriminativity this preprocessing step is quadratic. Obviously, when the data fit into main memory this preprocessing can be easily solved.

Algorithm 1 Frequency and Area-based Pattern Sampling

Input: a dataset D over ground set I,
a number of required patterns k,
an interestingness measure q
Output: k random sets of items, with each set $R \sim q(\mathcal{P}(I))$
1: calculate weights for each record $\boldsymbol{x} \in D$ according to q
2: Out $\leftarrow \emptyset$
3: **for** $i \leftarrow 1$ **to** k **do**
4: draw a record $\boldsymbol{x} \sim w(D)$
5: draw a subset $R \subseteq \boldsymbol{x}$ according to q
6: Out $\leftarrow$ Out $\cup \{R\}$
7: **end for**
8: **return** Out

The pseudocode of the direct sampling algorithm for the frequency and area measure is shown in Algorithm 1. For the frequency measure, the weight of data record $\boldsymbol{x} \in D$ is defined as $w(x) = 2^{|\boldsymbol{x}|}$, while the subset of the data record is drawn with uniform probability over all the subset, i.e. $R \sim u(\mathcal{P}(\boldsymbol{x}))$. In case that the area measure is used, the weights are defined as: $w(x) = |D|2^{|D|-1}$, and the subsets from the records are sampled by first determining the size of the subset with weights $1, \ldots, |\boldsymbol{x}|$ and then sample with uniform probability over all subsets of the previously determined size. A detailed description of the direct sampling algorithm for the discriminativity measure is given in Algorithm 2. Lines 1–3 of the algorithm describe the computation of the weights, while the actual sampling of the patterns is described at lines 5–6. A more detailed description of the direct sampling method, justification and correctness of the method is provided in [7].

Algorithm 2 Discriminativity-based Pattern Sampling

Input: datasets D^+, D^-, number of required patterns k
Output: k random sets of items, with each pattern $R \sim q_{\text{disc}}\mathcal{P}(I)$
1: **for all** $(\boldsymbol{x}^+, \boldsymbol{x}^-) \in D^+ \times D^-$ **do**
2: $w(\boldsymbol{x}^+, \boldsymbol{x}^-) = (2^{|\boldsymbol{x}^+ \setminus \boldsymbol{x}^-|} - 1)2^{|\boldsymbol{x}^+ \cap \boldsymbol{x}^-|}$
3: **end for**
4: Out$\leftarrow \emptyset$
5: **for** $i \leftarrow 1$ **to** k **do**
6: draw $(\boldsymbol{x}^+, \boldsymbol{x}^-) \sim w(D^+ \times D^-)$
7: draw $F \sim u(\mathcal{P}((\boldsymbol{x}^+ \setminus \boldsymbol{x}^-) \setminus \emptyset))$ and
$F' \sim u(\mathcal{P}(\boldsymbol{x}^+ \cap \boldsymbol{x}^-))$
8: $R \leftarrow F \cup F'$
9: Out $\leftarrow$ Out $\cup \{R\}$
10: **end for**
11: **return** Out

3.3 Distributed Pattern Sampling Algorithm

In this section we describe the adaptation of sequential sampling algorithms into the MapReduce framework. First, we discuss a method to efficiently draw k sample in one pass over the data, then we combine the different algorithms in the MapReduce framework. The major obstruction for applying Algorithm 1 and 2 to large datasets is the memory requirements of the pre-processing step, i.e. the computation of the weights for every record in the dataset. In particular, under the realistic assumption that $k \ll |D|$ in big data setting, the sampling of subsets from k records can be easily performed into main memory, independent of the size of the dataset. Moreover, the

computational complexity of this last step is equivalent to $O(k \cdot |I|)$ [7], and hence can be smoothly computed in a sequential setting.

Given the previous observations, the most straightforward approach to adapt the sequential algorithms would be the following procedure:

1. Divide the records/tuples over the m available nodes on the cluster.
2. For each node compute the weights of the records/tuples assigned to it, and sent over the partial results (weights + link where the record is stored) to a central node.
3. These results are then combined, followed by sampling k records/tuples from this index file.
4. Finally a sample is taken to obtain the desired patterns from these k records/tuples.

Although this approach is pretty simple, it has some serious drawbacks. First, the size of the index file, i.e. the resulting weights and the link where the data is stored, is dependent on the input size of the transaction database. Hence, for large dataset it is unreasonable to assume that it would fit into main memory. The resulting solution would require additional disk access and involves multiple communication steps between the nodes. A second disadvantage is that this approach would require multiple times access to the transaction database.

In this thesis, we resolve these issues by using a techniques developed to derive k weighted samples from streaming data [18]. In particular, whenever a data element (either a tuple or a data record) arrives, the A-RES [18] algorithm determines whether this data element is stored in a reservoir of size k or not. After the scanning of the complete transaction database, the reservoir holds the k selected data elements. From this point on, the pattern sampling can be further solved pretty straightforward.

Algorithm 3 A-RES: Reservoir Sampling

Input: A population $V = \{v_1, \ldots, v_n\}$ with corresponding weights $\{w_1, \ldots, w_n\}$
Output: a weighted random sample of V without replacement of size k

1: $R \leftarrow \emptyset$
2: insert the first k items in R and calculate their keys,
   i.e., $key_i \leftarrow r_i^{1/w_i}$ for $i = 1 \ldots k$ where r_i is a random number in $[0, 1]$
3: **for** $i \leftarrow k + 1$, **to** n **do**
4: $t \leftarrow$ the smallest key in R
5: $l_i \leftarrow r_i^{1/w_i}$, with $r_i \sim u(0, 1)$
6: **if** $l_i > t$ **then**
7: replace t and its corresponding item in R with respectively l_i and v_i
8: **end if**
9: **end for**
10: **return** the items in R

The A-RES [18] algorithm to sample k weighted items in one pass over the data, is a so called "reservoir" based approach. The pseudocode is given in Algorithm 3. The first k data elements and their corresponding keys are inserted into the reservoir. Then for every following data element its corresponding key is computed (line 4). A data element in the reservoir is replaced with a new one, whenever its corresponding key is lower then the key of the new data element (line 6–7). As such, after the first k data elements are processed, the reservoir contains at each moment the k data elements that have the largest key. Note that, for two keys $v_i = r_i^{1/w_i}$ and $v_j = r_j^{1/w_j}$ it holds that: $P[v_j \leq v_i] = w_j/(w_i + w_j)$.

In its basic setting the A-RES algorithm requires the generation of n random numbers, were n is the number of weighted items to sample from. In our case this is equal to number of data elements, i.e. depending on the interestingness measure either $|D|$ or $|D^+ \times D^-|$. This can be optimized

by making use of so called sampling with jumps. The main idea behind sampling with jumps is that a random variable determines what will be the next data element to enter the reservoir. A detailed description is provided in Algorithm 4. The crucial step consists of skipping items until the sum of their weights is larger than the smallest key in the reservoir (line 6 – 12). With this adjustment, and under the assumption that the weights are independent random variables, the number of random numbers generated is reduced from n to $k \cdot \log(n/k)$ [18].

Algorithm 4 AJ-RES

Input: A population $V = \{v_1, \ldots, v_n\}$ with corresponding weights $\{w_1, \ldots, w_n\}$
Output: a weighted random sample of V without replacement of size k
1: insert the first k items in R and calculate their keys
2: $t \leftarrow$ the smallest key in R
3: $r \leftarrow u(0, 1)$
4: $s \leftarrow 0$
5: **for** $i \leftarrow k + 1$ **to** n **do**
6: **if** $s > \log r / \log t$ **then**
7: $r_i \leftarrow u(t^{w_i}, 1)$
8: $l_i \leftarrow r_i^{1/w_i}$
9: replace t and its corresponding item in R with respectively l_i and v_i
10: $t \leftarrow$ the smallest key in R
11: $s \leftarrow 0$
12: **end if**
13: $s \leftarrow s + w_i$
14: **end for**
15: **return** the items in R

To use Algorithms 3 and 4 in the adjusted direct pattern sampling approach, we need to make a slight modification to the weighted sampling over a stream methods. That is, in the current setting, these algorithms sample k items without replacement while the direct sampling requires sampling with replacement. This obstacle can be trivially overcome by running k instances of the A-RES/AJ-RES with reservoir size 1. Note that, this adjustment requires that for each data element a decision has to be made for each of the k reservoirs. As a result, the number of random numbers to generate in the A-RES/AJ-RES algorithm increases respectively from n to kn, and from $k \log(n/k)$ to $k \log(n)$.

3.3.1 MapReduce Implementation

In order to use the MapReduce framework, the algorithm must be decomposed into map and reduce steps. Map functions are run in parallel on different parts of the input. The output of these map function consists of key-value pairs and is passed on to one or more reducers. The reduce step processes unique keys, and applies a user defined function to produce the final output of the MapReduce algorithm.

For the frequency and area-based sampling algorithm, i.e. the algorithm with linear pre-processing time (Algorithm 1), the adaptation into the MapReduce framework is given the precious algorithms now pretty straightforward. The pseudocode of these linear pre-processing time algorithm is given in Algorithm 5. In particular, each mapper draws according to the interestingness measure k records from its part of the transaction database (line 3), this is done by using Algorithm 3. Given that there are m mappers, the total number of sampled records equals mk. The reduce step selects from the overall output, i.e. the k records, with the largest key, this step is accomplished by one reducer. Finally, the reducer continues with sampling the k patterns from the selected records, as is done in Algorithm 1 (line 4). Once that the data is divided over the mappers, the required communication cost are low: only the output of the m mappers must be

send over the network. This algorithm requires that each mapper and the reducer can store k data records in main memory. Hence the memory requirements of the frequency and area based direct sampling algorithm are only dependent on the number of patterns of interest and the size of the largest transaction number, which is $O(k)$.

Algorithm 5 Distributed Frequency and Area-based Pattern Sampling

Input: a dataset D, a number of required patterns k, an interestingness measure q
Output: k random sets, with each set $R \sim q(\mathcal{P}(I))$

MapFunction
1: **for** $i \leftarrow 1$ **to** k **do**
2: run Algorithm 4, with sample size =1 and weights according to q
3: **end for**
4: **return** k (key,record) pairs
ReduceFunction
5: select the k pairs with the largest key
6: **for** $i \leftarrow 1,$ **to** k **do**
7: Out $\leftarrow$ Out $\cup\{$subsample from record$_i$ according to $q\}$
8: **end for**
9: **return** Out

Direct pattern sampling with the discriminativity measure of interest is however more problematic to integrate into the MapReduce framework. A major obstacle in this setting is to generate all tuples of records. That is, for every record from D^+ all records of D^- must be scanned. Preliminary experiments indicated that this adaptation was timewisely not doable for moderate datasets on the cluster we used. However, the amount of tuples to be generated can be drastically reduced. In particular, for a tuple $(\boldsymbol{x}^+, \boldsymbol{x}^-) \in D^+ \times D^-$ a weight $w(\boldsymbol{x}^+, \boldsymbol{x}^-) = (2^{|\boldsymbol{x}^+ \setminus \boldsymbol{x}^-|} - 1)2^{|\boldsymbol{x}^+ \cap \boldsymbol{x}^-|}$ is computed that determines with which probability that this tuples is sampled. However, a trivial upper bound of the weight—that is only dependent on $\boldsymbol{x}^+$—is $2^{|\boldsymbol{x}^+|}$. Hence, in the reservoir based approach (Algorithm 3) this upper bound can be used as a priori to determine that the tuple will not be inserted into the reservoir. And as a result, the algorithm can skip the data record $\boldsymbol{x}^-$. If however, the upper bound is larger than the minimum key in the reservoir, then the record should be scanned and the exact key should be determined. However, a downside of this approach is that the AJ-RES algorithm cannot be used to sample the k records. This is because the algorithm with jumps requires that every weight is computed. Therefore, the A-RES algorithm (Algorithm 3) to sample k elements from the input is used. A high level description of the MapReduce direct pattern sampling approach with discriminativity as interestingness measure is provided in Algorithm 6. Notice that, also in this case the memory requirements are only dependent on the number of patterns and the maximal transaction length.

Algorithm 6 Distributed Discriminativity-based Pattern Sampling

Input: datasets D^+ and D^-, a number of required patterns k
Output: k random sets, with each set $R \sim q_{\text{disc}}(\mathcal{P}(I))$

MapFunction
1: **for** $i \leftarrow 1$**to** $|D^-|$ **do**
2: **if** $r^{1/2^{|\boldsymbol{x}^+|}} >$ the lowest key in reservoir, with $r \leftarrow u(0,1)$ **then**
3: $\boldsymbol{x}^- \leftarrow$ next record from D^-
4: run Algorithm 3, sample size =1, weights according to q_{disc}
5: **else**
6: skip record in D^-
7: **end if**
8: **end for**
9: **return** k (key,record) pairs

ReduceFunction
10: select the k pairs with the largest key
11: **for** $i \leftarrow 1,$**to** k **do**
12: Out $\leftarrow$ Out $\cup\{$subsample from record$_i\}$
13: **end for**
14: **return** Out

A further important (implementation) issue is the data type to use for real numbers. In particular, the required arithmetic precision is dependent on the number of items in a record. That is, both Algorithm 5 and Algorithm 6 require the computation of $2^{|\boldsymbol{x}|}$. In order to be applicable to datasets that contain large transactions (e.g. $|\boldsymbol{x}| > 64$), specialized arbitrary precision arithmetic libraries are needed. In our implementation we used as default the native Java double representation.

3.4 Experimental Evaluation

In this section we experimentally evaluate the MapReduce implementations of the direct sampling approach. In particular, we conduct experiments with Algorithm 5 and the frequency interestingness measures, and with Algorithm 6 to derive discriminating patterns. We did not perform separated tests for Algorithm 5 with the area measure. This is because there is no fundamental computational difference between the two flavors of the same algorithm, and hence the computational results of one variant is also representative for the other variant.

The overall goal of the experimental evaluation is to examine whether the proposed algorithms are suited to mine large datasets. In particular, the main question to investigate is: given enough computational resources, are the algorithms able to extract a limited number of patterns on arbitrary large datasets? Although, it is not feasible to run the algorithm on arbitrary large datasets, we test the scalability by measuring the speedup. That is, how much faster is the algorithm when more computing units are added? The underlying idea is, if the speedup is good, arbitrary large datasets can be processed by adding a sufficient amount of nodes to the cluster. Another important characteristic of our algorithm to evaluate is its dependency on the input parameter k, that is the number of patterns to mine. In particular, as stated in section 3.3, the number of random numbers to generate is linearly dependent on the k parameter, and moreover is independent of the number of mappers that is being used. Hence, in the worst case the computation time is negatively affected by a factor k. Finally, we compare our direct sampling implementation with the popular and publicly available MapReduce implementation of parallel FP-growth [29]. In particular, we will report the run time for different support thresholds and discuss the results.

The experiments were conducted on the SURFSara Hadoop cluster[1]. The cluster is deployed

[1] https://www.surfsara.nl/project/hadoop

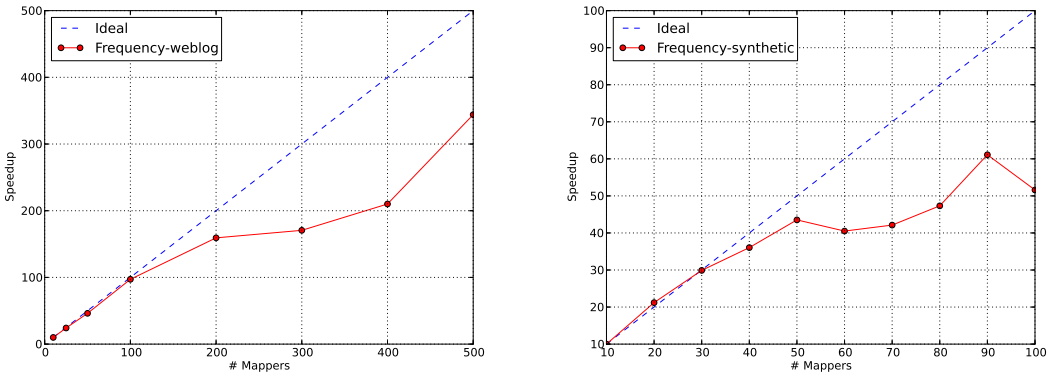


Figure 3.1: Speedup for the frequency based sampling for the Weblog dataset (left) and the synthetic dataset(right). The blue line indicates the ideal speedup, the red line shows the obtained speedup.

with Hadoop-0.20.2; the block size is 128M; the replication factor is 3; and the speculative task execution feature is disabled. This cluster consists of 82 nodes[2], and in each node there are respectively 8 slots of mappers and reduces, leading to totally 656 mappers and 656 reducers. However, since the cluster is shared with many users, and is being heavily used, the execution time of a job can vary quite a bit. In particular, if different I/O intensive jobs are scheduled in the same nodes, the different tasks can drastically reduce each other's performance. In order to obtain a better estimate on the scalability of the algorithms, we conducted all experiments three times, and report the average value.

3.4.1 Test Collection

Because of the high computational burden of the discriminativity-based sampling approach and the limited resources, we used smaller datasets for the discriminativity measure. For the frequency-based sampling, the following two datasets were used:

- **Weblog dataset** A web accesses log dataset from a popular Dutch portal website. Each visit to this website was recorded and enriched with derived features when available. The logs are taken over a period of two months and consists of more than 461M records (i.e. page views). The average record length is 22, while the maximum record length equals 30. In total, there are over $20M$ distinct items, while the overall size of the dataset is over 200GB

- **Synthetic dataset** An artificially dataset generated using the generator from the IBM Almaden Quest research group. This dataset contains 100M records (9.15GB) with an average length of 20, and a maximum of length of 55. The number of distinct items is equal to 1000.

For the discriminativity-based sampling two much smaller datasets were used for evaluating the performance:

- **Weblog one day dataset** A one day snapshot of the Weblog dataset. It contains 9.5M records, and its total size is 4.4GB. To split this dataset into two classes for the discriminativity-based sampling, we assumed that in 1% of the records the user clicked on one of the displayed links. The goal is to derive patterns that describes the different groups, i.e. users that click on a link and those that didn't.

[2]The cluster size may be increased in the future

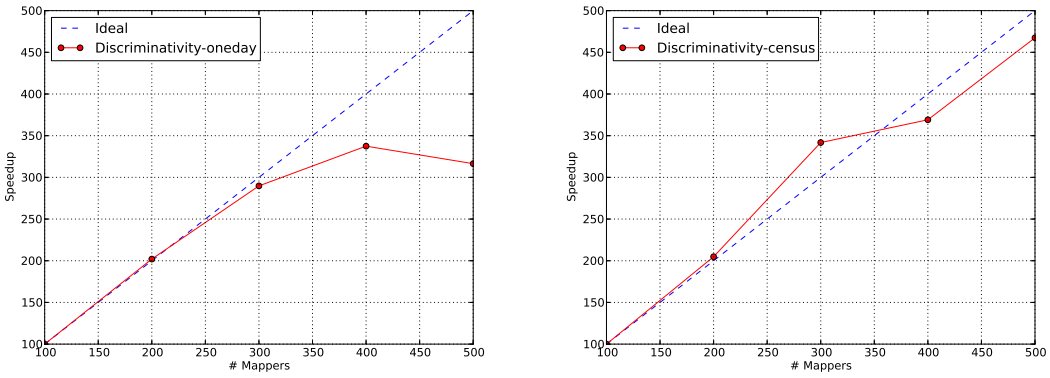


Figure 3.2: Speedup for discriminativity based sampling, the left plot contains the results obtained on the one day snapshot of the Weblog data, while the right one displays the results obtained on the census dataset. The blue line indicates the ideal speedup, the red line shows the obtained speedup.

- **Census income** [36] It is a dataset storing the income information with census data. We treat records with a salary larger than 50K as the positive ones, and the remaining as the negative ones. This leads to $18,568$ records in the positive set, and $280,717$ records in the negative set. All records are of equal length: 41.

3.4.2 Speedup Evaluation

In order to evaluate the speedup of our distributed sampling algorithms, we fixed the sample size and varied the number of mappers. Speedup refers to how many times a parallel algorithm is faster than the corresponding sequential algorithm, hence the optimal speedup that can be obtained is equivalent to the number of mappers used. In our experiments, however, we did not measure the running time of our sampling algorithms when using only one mapper due to the large sizes of the datasets. Instead, we took 10 mappers for the frequency-based sampling and 100 mappers for the discriminativity-based sampling as the baselines, and computed the speedup with respect to this baseline. In all experiments we fixed the number of patterns to sample to 100.

Frequency based sampling

Figure 3.1 shows the speedup results of the frequency-based sampling on the Weblog dataset (left plot) and the synthetic datasets (right plot). The speedup for the frequency based sampling on the Weblog data is close to the optimal speedup when the number of mappers is below 200. After this point, the overhead of involving more mappers and the limited computation time (489 seconds with 100 mappers), results in a less optimal speedup. This is also reflected when investigating the efficiency, defined as the speedup divided by the number of mappers, which drops from 0.97 with 100 mappers to 0.68 with 500 mappers. The overall computation time to sample 100 patterns from the Weblog datasets varies from $4,748$ seconds when 10 mappers are used till 138 seconds with 500 mappers. The same observations can be made for synthetic dataset, although for this dataset the ideal number of mappers is around 40. Obviously, this is because the synthetic dataset is relatively small, and the computation time is low. When more mappers are involved the efficiency drops from 0.9 with 40 mappers to 0.51 with 100 mappers. The computation time varies between 741 and 121 seconds. The speedup and efficiency obtained indicates that Algorithm 5 scales up pretty well for large datasets.

Discriminativity based sampling

Figure 3.2 shows the speedup results of the discriminativity based sampling on the one day snapshot of the Weblog dataset (left) and the census dataset (right). The speedup obtained on the Weblog snapshot is nearly ideal before the number of mappers reaches 300, but diminishes afterwards. The average running time of the baseline with 100 mappers is 2 hours and 25 minutes and drops to 46 minutes with 500 mappers. When 300 mappers are used the efficiency is 0.96, while for 500 mappers the efficiency is merely 0.63. The results are remarkably different for the census dataset. Although this dataset is considerably smaller, both the speedup and the efficiency are close to optimal when more mappers are involved. In particular, the running time drops from 77 minutes with 100 mappers to 16 minutes with 500 mappers, at the same time obtaining an efficiency of 0.93. A plausible explanation for this difference is that the transaction length of the census data is twice the average transaction length of the Weblog dataset, hence the computation of the weights requires more execution time in case of the census dataset. Another remarkable observation is that for the census data with 300 mappers, the so called super linear speedup is achieved. That is, the computation time with 300 mappers is less than three times the computation time with 100 mappers. However, this phenomenon is often observed when performing experimental evaluation of parallel algorithms and is likely caused by caching behavior of the nodes. Although the computation time is large for relatively small samples, the speedup and efficiency scores acquired suggests that discriminativity based sampling is able to derive interesting patterns from large datasets when enough computing resources are available. However, the number of mappers needed for large dataset is huge, and is likely to grow fastas the data size increases.

3.4.3 Scalability in the number of patterns

Since the sampling algorithms need to maintain as many reservoirs as the user defined sample size, the question that how heavily dependent the computation time is on this user specified parameterarises. In order to evaluate this, we fixed the number of mappers and varied the sample size. In particular, for both frequency as well as discriminativity based sampling we set the the number of mappers to 100, and varied the sample size between 100 and 800. The run time for the different sample sizes is compared with the time needed to sample 100 patterns, this ratio was taken and plotted versus the number of samples. Moreover, since the expected run time should roughly be linear to the size of samples, we drew a line indicating the expected ratio of the computation time needed. The results are displayed in Figure 3.3, left for the Weblog dataset with the frequency based sampling algorithm, right for census dataset with the discriminativity based sampling. The time needed for the frequency based sampling increases along approximately half as fast as the the sample size, while for the discriminativity based sampling the computation increases approximately at the same pace as the sample size. Note that for the later case, the experiments with $k = 300$ and $k = 400$ indicate that the computation time increases faster than the sample size. However, this is most likely caused because of the heavy workload on the cluster, resulting in a wide variety of time measurements for these experiments. For example, in the run with 300 samples the maximal difference between two runs was more than three hours on a total computation time of seven hours on average.

3.4.4 Comparison with parallel FP-growth

A publicly available MapReduce implementation of parallel FP-growth (PFP) [29] is available in the popular Apache Mahout[3] machine learning library. Although the setting in which PFP operates differs from the direct sampling approach, i.e. given a minimum support threshold PFP mines all patterns that satisfy this threshold and then reports the k requested patterns, we think that the comparison is interesting from an end user's point of view.

We run PFP, with various minimum support thresholds, on the Hadoop cluster with the number of mappers fixed to 100. The results are summarized in Table 3.1. Besides the run time,

[3] http://mahout.apache.org/

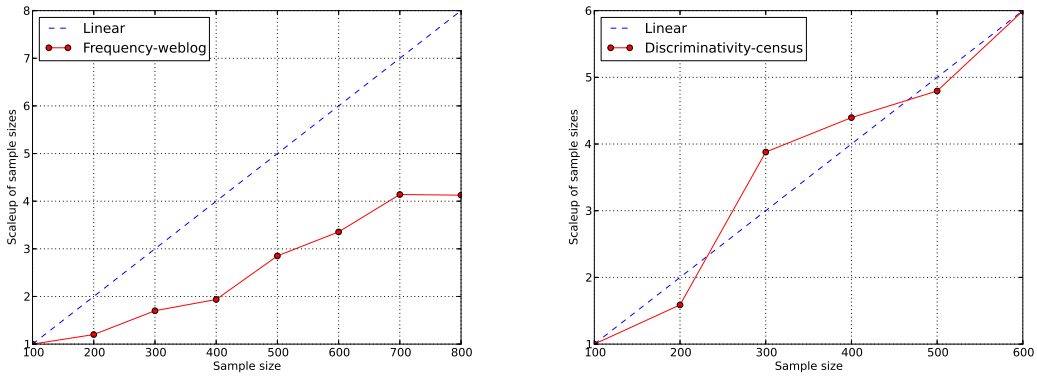


Figure 3.3: The ratio of the computation needed for different sample size and the baseline, plotted in red. Results for the Weblog dataset (left) with frequency based sampling and the census data with discriminativity based sampling (right). Additionally, the blue line displays the ratio expected.

Table 3.1: Run time, number of frequent items, and number of frequent itemsets obtained by PFP for the synthetic and the Weblog datasets. The results are shown for various minimum support thresholds.

	Minsup	Time	Singletons	Patterns
Synthetic	99%	406	0	0
	5.6%	1,518	63	63
Weblog	99%	4,118	3	3
	1%	5,625	128	4,543
	0.1%	6,325	304	10,825

we also show the number of frequent items, that is singleton itemsets, and the number of frequent patterns. Note that the number of frequent itemsets is limited by the user specified parameter k (in our case $k = 50$), that defines the maximum number of frequent itemsets to show per item.

The first noteworthy observation is that PFP needs far more computation time in its pre-processing step, i.e. counting the single items, than the frequency based sampling approach need to derive 100 patterns. In particular, 406 versus 144 seconds for the synthetic dataset and 4,118 vs. 489 for the Weblog dataset. Another issue is that for the synthetic dataset we were not able to mine 100 frequent patterns, we were only able to derive 63 frequent singletons. Lowering the frequency threshold resulted for this dataset in out of memory errors. For the Weblog dataset, we were able to obtain the desired number of patterns by lowering the frequency threshold, but the resulting computation time for PFP is multiple times the time needed for the frequency based sampling algorithm. Although, the results presented in Table 3.1 suggests that frequency based sampling approach is superior in terms of run time than PFP, there are many different parameters that influence the computation time. As we have previously shown, the computation time for frequency based sampling is both dependent on the number of samples and on the average transaction length. On the other hand, for PFP the computation time is dependent on the minimum support threshold and the density of the data. Provided that PFP does not run out of memory, it is likely that it in some settings PFP outperforms the frequency based sampling when a large number of patterns is demanded.

3.5 Related Work

Because the enumeration of all frequent patterns in a database is inherently a computational exhaustive task, parallelization of frequent itemset mining algorithms has been an active research topic for a while, see for example [45]. The most effort has been put into the so called shared memory parallelization, however this type of parallelization is not scalable for huge datasets, due to the extensive memory requirements. Parallel frequent itemset mining in distributed environments, has been examined in a few papers, see for example [19], which results in algorithms that are typically scalable up to a hundred different computers.

Recently, the focus of the research has been shifted towards mining algorithms in the MapReduce framework. The main motivation is to drastically increase the number of resources and to have fault tolerance mechanisms automatically incorporated into the system. Most related to our work is the MapReduce implementation of the popular FP-Growth [25] algorithm: parallel FP-growth (PFP). This algorithm is described in the work by Li et al. [29]. The main idea of PFP is to first group the transactions, and then build a local FP-Tree for each group. The outcome of these trees is aggregated, and the k most frequent patterns are reported. The major difference with our approach is that PFP performs an exhaustive search, although only the top k frequent items are reported. Another important difference is the amount of communication that is needed between mappers and reducers. In particular, the pre-processing step of PFP, i.e. the grouping, requires multiple communication steps. In Section 3.4 we experimentally compare PFP with our frequency based sampling approach.

3.6 Summary

In this chapter, we successfully casted the recently proposed algorithms of directly sampling local patterns [7] into the MapReduce framework. The theoretical analysis show that the memory requirements of both proposed algorithms are limited to the number of patterns in the output. Extensive experimental evaluation showed the scalability of the frequency based sampling approach, and hence its applicability for "big" data. Although, the discriminativity-based sampling scales reasonably well for small samples, it is doubtful whether this algorithm will scale for large dataset, mainly because its intensive usage of computational resources. For both approaches we showed that the computation time is roughly linear dependent on the number of patterns to sample, which makes the algorithm suitable in most practical settings. Furthermore, we compared the frequency based sampling with a state-of-the-art MapReduce implementation of parallel FP-growth, and showed that our approach outperforms PFP in term of computation time. Further research is involved with speeding up the quadratic pre-processing step needed for discriminativity based sampling. In particular, a recent follow-up paper by Boley et al. [6], addressed this problem and used coupling from the past to avoid the computational bottleneck, it is however still open how to integrate the coupling from the past technique into an MapReduce framework.

The source code is publicly available at: `http://www.win.tue.nl/~mpechen/projects/capa/`, and the work in this chapter has been published in BigMine '13: Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications.

Chapter 4

Classifying Skewed Data Streams Using Patterns

In our click prediction problem setting, pageviews come as a data stream. Formally, a data stream DS can be defined as a sequence of transactions, $DS = (T_0, T_1, \dots)$, where $T_i = \{\boldsymbol{x}_i, c_i, t_i\}$ is the i-th arrived transaction, in which $\boldsymbol{x}_i$ is a pageview, or an instance in the context of classification, $c_i \in \{0, 1\}$ is the click status label, and t_i is the time when the transaction arrives. The pageviews arrive at high speed, strictly constraining pre-processing algorithms in space and time. To adhere to these constraints, specific requirements have to be fulfilled by the stream processing algorithms [5]:

1. scan one example at a time, and inspect it at most once;
2. use a limited amount of memory;
3. work in a limited amount of time; and
4. be ready to predict at any time.

Besides these difficulties, the class distribution of the pageview data stream is highly skewed, since very few visitors click on the index page of a website. Classification models may classify most instances with a minority label as ones with a majority label.

In this chapter, we will first review the state-of-art techniques to cope with skewed data streams, then propose our solution to handle the pageview data stream, finally give possible extensions of the methods in our solution.

4.1 Related Work

Classifying the skewed pageview data streams using patterns involves three main challenges: handling skewness, mining patterns over data streams, and classifying data streams. Difficulties in each challenge and existing methods to overcome these difficulties are reviewed in this section.

4.1.1 Classification on Skewed Datasets

Skewed class distribution poses great challenges in model learning. In real application data, the positive instances are much less popular than the negative ones. Misclassifying a positive instance usually leads to a much higher loss compared to that of misclassifying a negative one. Traditional inductive learning methods would perform rather poorly on such datasets with skewed distribution, since the goal of those methods is usually to minimize classification error rate. As a result, the positive instances tend to be ignored and every instance would be predicted as negative. In such a case, we could build a model up to 99% accuracy, but of very little use in practice.

The skewed dataset classification problem has been well studied in recent years. A number of solutions to this problem were previously proposed at both the data and algorithmic level [9]. At the data level, these solutions include different forms of sampling to build a balanced dataset,

on which classification models will be subsequently trained. Two typical forms of sampling are random oversampling and random undersampling. In the oversampling approach [31], data from the minority class are duplicated at random until the imbalance is eliminated. While in the undersampling approach [28], data from the majority class are randomly eliminated to balance the classes.

There are several issues that the sampling approach is facing. Firstly, the desired class distribution for a learning algorithm often largely depends on the data. It has been observed that a naturally occurring distribution is not always the optimal distribution, neither a precisely balanced distribution [44]. Therefore, the sample size of oversampling or undersampling is usually empirically detected and set. Moreover, both undersampling and oversampling can lead to overfitting. In addition, undersampling involves a loss of information by discarding potentially important data. Finally, oversampling can introduce an additional computational task if the dataset is already fairly large but imbalanced [37]

At the algorithmic level, solutions include cost-sensitive learning, recognition-based learning (one-class learning) rather than discrimination-based learning, etc. Cost sensitive learning assigns different error penalties: an misclassification of an instance belonging to the minority class would have a larger penalty than one for the majority class. These penalties are represented as cost matrix, and often assumed known for different types of errors or even instances, which can be used at classification time [20].

Though it is often reported that cost-sensitive learning practically outperforms random sampling, such as in [9], this learning faces the following issues. Firstly, cost matrices are often unknown. Secondly, it can be argued that the effect of assessing penalties is equivalent to changing the relative data distribution in the two classes, or, in other words, to re-balancing the data. The experiments carried in [33] suggest that sampling, as well as adjusting the cost matrix, have the same effect as moving the decision threshold. Finally, cost-sensitive learning does not modify the class distribution of the data the way re-sampling does.

Besides sampling and cost-sensitive learning, another approach is recognition-based learning. A recognition-based learning approach provides an alternative to discrimination where the model is created based on the instances of the target class alone. Here, one attempts to measure (either implicitly or explicitly) the amount of similarity between an instance and the target class, where classification is accomplished by imposing a threshold on the similarity value [27]. Mainly, two classes of learners were previously studied in the context of the recognition-based one-class approach—SVMs [39, 41] and autoencoders [34, 27] —and were found to be competitive [34].

4.1.2 Classification over Data Streams

The major research challenges in data stream mining are represented in concept drifting, resource adaptivity, high data rates, and the unbounded memory requirements. There are many well established approaches to solve issues in streaming data classification. These approaches can broadly categorized as data-based and task-based [1]. In data-based solutions, the idea is to examine only a subset of the whole data set or to transform the data vertically or horizontally to an approximate smaller size data representation. In [1], data-based solutions include the following.

- Sampling: probabilistically choosing a data item to be processed or not. Boundaries of the error rate of the computation are given as a function of the sampling rate.
- Load shedding: ignoring a chunk of data, or dropping a sequence of data streams.
- Sketching: the process of randomly projecting a subset of the features of the incoming data.
- Aggregation: computing statistical measures such as means as variance that summarize the incoming stream.

Such an approach allows us to utilize many known data mining techniques to the case of data streams.

On the other hand, in task-based solutions, some standard algorithmic modification techniques can be used to achieve time and space efficient solutions [21]. Task-based techniques are those methods that modify existing techniques or invent new ones in order to address the computational

challenges of data stream processing. A typical solution is the sliding window technique, which detailedly analyzing the most recent data items and summarized versions of the old ones, assuming that users are more concerned with the analysis of most recent data streams [21].

4.1.3 Pattern Mining over Data Streams

In Chapter 2 we introduced pattern mining (frequent itemset mining), and in Chapter 3 we described an algorithm that samples patterns of interest from large datasets. However, in the streaming setting, mining desired patterns from data streams still needs exploration.

Many efficient frequent pattern mining algorithms that have been developed in the last decade typically require datasets to be stored in persistent storage and involve two or more passes over the dataset. Recently, there has been much interest in data arriving in the form of continuous and infinite data streams.

In the recent years, several new mining algorithms have been proposed to find frequent patterns over data streams. Usually mining frequent patterns from data streams relies on window models, where a window is a subsequence between i-th and j-th arrived transactions, denoted by $W[i,j] = (t_i, t_{i+1}, \ldots, t_j), i < j$. Different types of windows are summarized below.

Landmark window In this model, frequent patterns are mined from a starting time point i to the current time point t, i.e., over the window $W[i,t]$. A special case of the landmark window is when $i = 1$, in which frequent itemsets are mined over the entire data stream.

Sliding window In many cases, recent data may have a greater importance. This motivates the focus on the recent frequent patterns in a past window $W[t-w+1,t]$ where w is the window length and t is the current time point. As time changes, the window will keep its size and *slide* along with the current time point.

Damped window model This model assigns more weights to the recently arrived transactions. A simple way to do that is to define a decay rate [8], use this rate to update the previously arrived transactions (by multiplication) as a new transaction arrives. Correspondingly, the count of an itemset is also defined based on the weight of each transaction.

4.2 Our Approach

To deal with the pageview data stream, we adopt the sliding window technique. At a time point, we maintain a window with a fixed size, build pattern-based classification models within this window, and then apply the built models to the next coming stream. The window slides when the coming stream fulfills a user-specific size. Formally, at a given time point t, we maintain a window $W[t-w+1,t]$ consisting of transactions, where w is a user-specific window size, or time period. A transaction consists of the time point that this transaction happens, a pageview (or an instance in the context of classification), and a label if the click status of the pageview is known.

In a window of transactions, we use *1)* undersampling to deal with the class distribution skewness, *2)* distributed pattern sampling described in Chapter 3 to mine discriminative patterns, *3)* pattern post-processing to select useful patterns for classification, *4)* instance encoding to represent instances, *5)* and the encoded instances to build classification models. Then the labels of the coming data stream can be predicted by the built classification models.

4.2.1 Undersampling

To handle the skewed class distribution of the dataset in a window, we adopt the sampling technique to draw undersamples of both the positives and the negatives. Cost-sensitive learning is not applied to our problem since it needs a predefined cost-matrix, which we cannot provide.

Algorithm 7 shows our straightforward sampling method. For a positive training set D^+ and a negative set D^- where $\mid D^+ \mid \ll \mid D^- \mid$, we draw n instances from the positive set and negative set respectively, where n is a user-defined parameter that cannot exceed $\mid D^+ \mid$. One caveat is the choice of n: in an ideal case, n should be as large as $\mid D^+ \mid$; however, a large n may lead to additional computational cost for both pattern mining and pattern post-processing. Therefore, the value of n should be carefully chosen, as a compromise of time cost and effectiveness.

Algorithm 7 Dataset Undersampling

Input: A set of positive training instances D^+,
a set of negative training instances D^-,
a user-specific number n
Output: An balanced dataset
1: $P' \leftarrow$sample n instances from D^+
2: $N' \leftarrow$sample n instances from D^-
3: **return** $P' \cup N'$

4.2.2 Pattern Mining

After undersampling from a window, a balanced dataset D is obtained with n positive and negative instances respectively. Then we mine k discriminative patterns for the positive class from D using the distributed pattern sampling algorithm. The k mined patterns need to be post-processed before using them in classification.

Pattern Post-processing

Mined patterns need post-processing because not every one can contribute to the improvement of classification. Firstly, patterns with low supports can still be sampled due to the essence of sampling, and these patterns often only cover a very small number of instances. For an extreme example, suppose a sampled pattern only exists in one instance of class positive, then the only contribution of this pattern in classification is identifying that positive instance, which is of little use in practice. Secondly, patterns with high supports but not discriminative can be sampled as well. A pattern exists almost in all instances may be sampled, yet this pattern is unlikely to be capable of differentiating between classes, since given an instance represented by this pattern, the probabilities of being positive and of being negative may be equal. Therefore, not every pattern is equally useful; and post-processing them is necessary.

Our post-processing of patterns focuses on increasing generalities of patterns, and retaining non-redundant patterns with high discriminative powers. It consists of two steps: pattern shortening and pattern selection, which address these two issues respectively.

Figure 4.1 illustrates the idea of the two-step post-processing procedure. In each subfigure, the light-blued (upper) part of the rectangle represents the positives in the dataset, and the lower part represents the negatives in the dataset. Each vertical bar represents a pattern. The support of a pattern is indicated by the length of the corresponding vertical bar. One can imagine that in a rectangle, instances are piled vertically, so that a pattern has a larger support if its corresponding vertical bar covers more in the rectangle. The discriminative power of a pattern can be seen as the proportion of the positive area to the negative area that the corresponding vertical bar locates in. Note that one pattern cannot naturally only cover consecutive instances; nevertheless, it is presented so to exemplify our idea.

The post-processing begins with shortening the mine patterns shown in 4.1(a). We attempt to increase the support of eah pattern by removing items from it, while retaining its discriminative power. For a pattern, there must a subset of it which has a maximal support while having at least the same discriminative power of this pattern. We call such a subset the target of a pattern. We search for the target of a pattern by random local walks—randomly removing items from the pattern. Figure 4.1(b) shows the search attempts— vertical bars are trying to become longer ones

and stay at the same part of the rectangle, as patterns are trying to cover more instances and retain at least the same discriminative power. The targets of patterns are usually not the same; and the targets may not be mined by our pattern mining algorithm, which are represented with dashed lines in 4.1(b). Targets sometimes are not reached, patterns may stay where they are, or just half-way of the path to their targets. We assume that all patterns can find their targets in this example. Figure 4.1(c) shows the shortend patterns.

The second step of the post-processing is pattern selection, which selects patterns distant from each other but have high discriminative powers. It starts with selecting the most discriminative pattern, as the dashed line labeled 1 in Figure 4.1(d). Then, a distant pattern from the previously selected one yet with a high discriminative power is selected, as indicated by the dashed line labeled 2. Distance is intuitively measured by the number of instances that a collection of patterns can cover. For example, if pattern α_1 and pattern α_2 can cover 10 instances, and pattern α_1 and pattern α_3 can cover 4 instances, then we say α_2 is more distant from α_1 than α_3 is. This distance can be visualized by the length of the bar horizontally projected from patterns. We continue in this fashion until the whole dataset can be covered by the selected patterns. The finally selected patterns are the four shown in Figure 4.1(d).

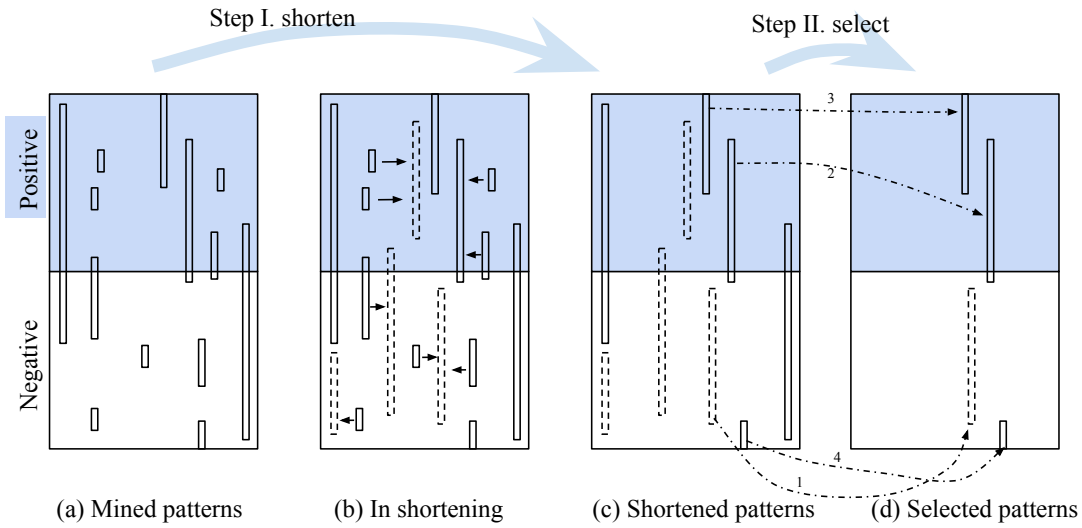


Figure 4.1: Illustration of post-processing

The formal description of the two-step post-processing procedure is as follows.

Pattern shortening In pattern shortening, patterns are shortened while retaining their discriminative power, thereby implicitly increasing their support and generalization. It is done essentially by performing a local random search —randomly remove items in a pattern without losing the discriminative power of this pattern. Algorithm 8 shows this procedure. For each pattern p in a given pattern collection P, and a given binary labeled dataset D, we firstly calculate its discriminative quality $f_0 = f(p)$, where f is a discriminative measure, e.g. information gain. Then we randomly remove items from p, while $f(p) \geq f_0$.

Algorithm 8 Pattern Shortening

Input: a labeled dataset D,
a pattern collection P,
and a discriminative measure function f
Output: a shortened pattern collection S

```
1: S ← ∅
2: for each pattern p(i_1, i_2, ...) in P do
3:    f_0 ← f(p)
4:    while f(p) ≥ f_0 do
5:       p' ← p
6:       remove a random item in p
7:       S ← S ∩ {p'}
8:    end while
9: end for
10: return S
```

Pattern selection The purpose of pattern selection is to select a small number of distant patterns yet with high discriminative power, where the distance of two patterns is the cosine distance of them, as shown in Equation 4.1. Both the pattern sampling and the pattern shortening may introduce redundant patterns, which differentiate between same subsets of the dataset. Furthermore, it is possible that the result of pattern sampling contains non-discriminative patterns, which do not help classification but are noisy.

Intuitively, we can rank patterns in the descending order of their discriminative power, and select those at top. In this way, however, redundancy is not considered. To avoid selecting redundant patterns, we add the distances between patterns.

Algorithm 9 shows the pattern selection procedure. For a training set D and a pattern collection P, we iteratively apply pattern selection step. At each step, we consider not only the discriminative power of a pattern, but also its distance to the selected patterns, because redundant patterns should not be included in our selection results. After selecting a pattern, all the training instances containing this pattern are eliminated from D and this pattern is marked as selected. So that in the next iteration, the same step is applied on a smaller set of training instances. This algorithm continues in an iterative fashion until either all the training instances are eliminated (or selected patterns entirely cover D several times) or all the patterns are selected. The discriminative measure function f can be information gain [38], which measures how well a pattern can separate the dataset. The redundancy function d is the cosine distance between two patterns. Specifically, for two patterns α_1 and α_2, the cosine distance is

$$d(\alpha_1, \alpha_2) = \frac{\mid \alpha_1 \cap \alpha_2 \mid}{\mid \alpha_1 \mid \cdot \mid \alpha_2 \mid}. \tag{4.1}$$

The number of selected patterns is determined by a dataset coverage constraint threshold δ, as in [30]. The coverage threshold δ is set to ensure that each training instance is covered at least δ times by the selected patterns. In this way, the number of patterns selected is automatically determined, given a user-specified parameter δ.

Algorithm 9 Pattern Selection

Input: a labeled dataset D, a pattern collection P,
a discriminative measure function f,
a redundancy measure function d,
a dataset coverage threshold δ
Output: a small pattern collection S, with high discriminative power

```
S ← ∅
D_swp ← D
while D is not empty, or P is not empty do
    s ← arg max_{x∈P\S}(f(x) · min_{y∈S} d(x, y)),
    if s can cover at least one instance in D then
        S ← S ∪ {s}
    end if
    F ← F \ {s}
    D ← D \ D_s
    if not D and δ > 0 then
        D ← D_swp
        δ ← δ − 1
    end if
end while
return S
```

To summarize, our pattern post-processing is: given a pattern collection P and a labeled dataset D, we shorten each pattern in P by applying Algorithm 8, and then select a subset S of P by applying Algorithm 9 with a user-specific dataset coverage threshold δ.

4.2.3 Using Patterns in Classification

Instances are encoded with patterns before inputting them into classification models. Each pattern forms a binary attribute to an instance, with its value equal to 1 if the pattern exists in the instance, or 0 otherwise. There are two ways of encoding: either only considering the patterns, or augmenting original attributes are considered. Formally, given a selected pattern collection P and an instance $\boldsymbol{x} = \{i_1, i_2, \ldots, i_m\}$,

- either replace $\boldsymbol{x}$ by binary attributes corresponding to the elements of $p_1, \ldots, p_k \in P$, leading to a new instance $\boldsymbol{x}' = \{i'_1, \ldots, i'_k\}$ with k attributes:

$$i'_j = \begin{cases} 1, & \text{if } p_j \subseteq \boldsymbol{x}; \\ 0, & \text{otherwise} \end{cases}$$

  , where $j = 1 \ldots k$.

- or augment $\boldsymbol{x}$ by binary attributes corresponding to the elements of $p_1, \ldots, p_k \in P$, leading to a new instance $\boldsymbol{x}'' = \{i''_1, \ldots, i''_{m+k}\}$ with $m + k$ attributes:

$$i''_j = \begin{cases} i_j, & \text{if } j \leq m; \\ 1, & \text{if } j > m \text{ and } p_{j-m} \subseteq \boldsymbol{x}; \\ 0, & \text{otherwise.} \end{cases}$$

  where $j = 1 \ldots m + k$.

For each instance in a window or in the coming data stream, we encode it by one of these two ways with selected patterns, before building or testing classifiers.

4.2.4 Classification Models

Encoded instances are essentially vectors again, and various existing classification models can be trained on these instances. Two typical ones are Naive Bayes and the C4.5 decision tree.

Trained models can be applied to classifying the coming transactions in the data stream.

4.3 Possible Extensions

A framework that utilizes patterns in skewed data stream classification was described in the last section. This framework employs very basic settings in each step, so that there is still room to extend this framework. In particular, we can adjust the undersampling strategy, update patterns, integrate pattern mining and post-processing, update classification models and utilize other classification models than Naive Bayes or decision trees.

4.3.1 Adjustment of Undersampling

In a sliding window, a balanced training dataset is undersampled from the data within this window. The quantities of the positives and the negatives in this balanced dataset are exactly equal.

However, this exactly balanced distribution of class might not be the optimal for building classification models [44]. Classification models trained on a slightly imbalanced training set may have better performances.

We can adapt the balance level of a training set by setting a balance ratio r when undersampling from negative instances. That is, instead of sampling a same amount of negative instances to positive ones, we sample n_p/r negative instances, where n_p is the number of the positive instances. r is typically between 0.3 to 0.6 to make the distribution balanced [22].

4.3.2 Pattern Mining

After a training set has been undersampled from the data within a sliding window, we mine patters of interest from this set and then post-process them.

Patterns will be completely re-mined once the window slides. An alternative strategy may be updating the mined the patterns after the window has slided. In this way useful patterns in the past may still have a chance to be retained and contribute to classification.

Another modification in our approach would be the integration of pattern mining and post-processing. such that the post-processing is parallelized along with pattern mining to benefit from the distribution.

These two modifications are described in the following.

Pattern Updating

In our approach, we mine discriminative patterns from a undersampled training set from a window on a data stream. The patterns are totally replaced by lately mined ones when the window slides. However, some old patterns may still be important in the future; removing them without any consideration seems careless. One way to take old patterns into account is reducing their importance rather than simply replacing them.

Motivated by [8], a decay rate d is applied to reduce the weights of mined patterns for a fixed decay unit. A decay unit determines the chunk of information to be decayed together; in our case, this unit is the step size of a sliding window. A decay rate is defined by two parameters: a decay-base b and a decay-base-life h. A decay-base b determines the amount of weight reduction per a decay-unit and it is greater than 1. When the weight of the current information is set to 1, a decay-base-life h is defined by the number of decay-units that makes the current weight be b^{-1}. Based on these two parameters, a decay rate d is defined as follows:

$$d = b^{-1/h} \qquad (b > 1, h \geq 1, b^{-1} \leq d < 1) \tag{4.2}$$

Recall our two-step distributed pattern sampling algorithm, each discriminative pattern is drawn from two instances $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$. And these two instances are drawn according to the weight defined by $l = r^{1/w}$, where $r \in [0, 1]$ is a uniform random number and $w(\boldsymbol{x}^+, \boldsymbol{x}^-) = (2^{|\boldsymbol{x}^+ \setminus \boldsymbol{x}^-|} - 1)2^{|\boldsymbol{x}^+ \cap \boldsymbol{x}^-|}$. We can continue use this weight l as the pattern's weight.

With a mined pattern collection P, as the sliding window moves, the weight of each pattern is multiplied by d, i.e. $w(p_i) = d \cdot w(p_i)$ for $p_i \in P$. Then we mine k new discriminative patterns P', and update our patterns for classification (post-processing) as the patterns with top-k weights in $P \cup P'$.

Integration of Pattern Mining and Post-processing

For a training set undersampled within a sliding window, pattern mining and pattern post-processing are two separate steps, i.e. patterns are firstly mined, then post-processed. Our proposed pattern sampling algorithms in Chapter 3 are designed to mine patterns from large datasets in parallel, and are indeed implemented in a distributed computing programming framework.

Post-processing, on the other hand, is a sequential algorithm, which can run only in a single machine with limited main memory. Even worse, computing the discriminative measure, e.g. information gain, of a single pattern needs multiple scan of the dataset. In particular, pattern shortening needs at least $k \cdot N$ scans of the dataset and pattern selection needs at least one scan of the dataset (though the scans in pattern selection can be avoided by caching the measures from pattern shortening), where k is the number of mined patterns and N is the number of the instances in the dataset. This is very likely to be a bottleneck of our approach in terms of running time.

It would be very nice that post-processing can be fit into the MapReduce framework in order to benefit the existing computing resources. A tentative idea is that, pattern shortening can be straightforwardly parallelized by partitioning the pattern sample and shortening the patterns in each partition in parallel. The drawback of this way is that, the dataset needs to be copied into each computing node, since determining the discriminative measures of patterns needs whole scans of the dataset. Nevertheless, it might still be faster than shortening patterns in a sequential fashion.

Another better tentative proposal is to post-process patterns while mining them. In the first step of our distributed pattern sampling algorithm for the discriminativity interestingness measure, data tuples are drew in each node; it might be possible that post-processing a data tuple immediately after this data tuple has been drew. However, parallelizing post-processing is beneficial but still open in this thesis.

4.3.3 Classification Models

In our approach, we rebuild classifiers once the window slides. This is not a truly streaming fashion. Instead, a better way is to update the classifier once the window slides.

Anohter potential improvement is to use ensemble classifiers to boost classification performance.

Classifier Updating

We rebuild classifiers with each slide of the window. In this way valuable prior information is abandoned. Alternatively, we can keep this information within models by updating them. Specifically, suppose we have a trained model m_t at time point t, then Δt time unit passes and the window slides. A new training set D is undersampled, we update the model m_t with D, and this updated model is thus $m_{t'}$ where $t' = t + \Delta t$.

Many classification models are updatable, including Naive Bayes, VFDT [15] and logistic regression. However, one noteworthy point is the feature space may change due to the update of patterns. Nevertheless, there are existing works concerned with dynamic feature spaces, such as [35], which can be integrated into our framework.

Ensemble Classifiers

Single classifiers e.g. Naive Bayes and decision trees are used in our classification. An alternative way is to use ensemble classifiers.

The balanced dataset from undersampling can significantly improve the recall of the skewed positives, however, it also introduces variances to pattern mining and model learning. A classifier based on a single sample could increase false positive rate although it improves recall on the positives [11].

To reduce the variance caused by sampling, Cheng [11] proposed an ensemble classification approach, which generates multiple training sets with repeated samples of the positives and undersamples of the negatives. Multiple classifiers are built based on the different re-balanced samples and an ensemble is used to combine the output of different classifiers. This method can significantly reduce the false positive rate while maintaining a high recall rate.

Her method is shown in Algorithm 10. Given a positive set P and a negative set N where $|P| \ll |N|$. A series of undersamples $N_i i$, $i = 1, \ldots, k_d$ are undersampled from N, and N_i is comparable to P in size. The actual size of N_i can be decided by a user-specified parameter r, $|N_i| = |P|/r$. For each undersample set N_i and the positive set P, we apply pattern mining and feature selection. A classifier C_i is trained on $P \cup N_i$. For testing, decisions from k_d classifiers $C_1, C_2, \ldots, C_{k_d}$ are combined.

Algorithm 10 Balanced Data Ensemble

Input: A set of positive training instances P,
a set of negative training instances N,
ensemble number k_d , sampling ratio r,
and a test set T
Output: An ensemble for posterior probability estimate
1: Compute the number of negative samples by $|P|/r$
2: **for** $i \leftarrow 1$ to k_d **do**
3: Draw a sample N_i from N
4: Mine frequent patterns from P and N
5: Train a classifier based on $P \cup N_i$
6: Estimate posterior probability $\{f^E(x)\}_{x \in T}$
7: **end for**
8: Estimate posterior probability $\{f^E(x)\}_{x \in T}$ by combining ensemble outputs.

4.4 Summary

In this chapter, we described the challenges of our click prediction problem, and categorized these challenges into the skewed class distribution, mining patterns from data streams, and classification over data streams. State-of-art methods to handle these challenges were also briefly reviewed.

In the particular case of our click prediction problem, we propose to adopt a sliding window in order to handle the streaming data, undersample negatives to build balanced datasets, encode instances with post-processed patterns, and train models on the encoded instances in the sliding window to predict coming pageviews.

Additionally, several improvements of our methods were also proposed, including adjusting sample sizes of both positives and negatives, updating patterns as the sliding window moves, parallelizing pattern post-processing, updating classifiers and replace single classifiers by ensemble ones.

In conclusion, a combination of methods for effectively classifying the pageviews are presented in this chapter.

Chapter 5

Case Study: A Dutch Portal Website

5.1 Introduction

This case study is concerned with a Dutch portal website. This website has a large number of links divided into sections on its index page. Though for a user, he/she can manually configure which sections to display, this website intends to automatically and dynamically adjust the placements of the sections according to the user's preference. For example, if a user prefers to weather information, the website can directly display that information in a conspicuous position on its index page. A very initial step to accomplish this task is to know the user's preference, or at least, his/her interests on this website, which is implied by his/her click on a link.

Predicting a user's click on the index page of this website conforms to our problem setting proposed in Section 1.1, and the challenges described hold here. In brief, the first difficulty is that little information of the user can be captured by the website. Unlike click prediction in search engines, this website is visited mostly directly from its index page, without any redirection from other sites; thus one kind of useful information, keyword, does not exist. The second difficulty is that the data comes as a streaming at a high velocity. For a concrete example, there are 46 views on this website per second.

In this chapter, we employ the methods discussed in previous chapters to explore the potentially shared patterns, build classifiers with those patterns, and predict whether a user would click on the index page. Before we describe the application of those methods, one extra step, pre-processing, is necessary to transform the raw data to classifiable instances. Subsequently, we experimentally validate our pattern-based classification approach in an offline setting, i.e. on a sample of the whole dataset. Finally we lift our approach to a simulating online setting and give conclusions.

In experiments, we focus on these issues:

1. In the offline setting, we examine the effects of discriminative patterns on classification performance.
2. In the offline setting, we explore the effects of different parameters used in our approach on classification performance; and find a set of parameters that best fit the data sample in terms of classification performance.
3. In the simulating online setting, we discover the improvement of classification performance upon classification methods without patterns, and then answer to what extent our approach is feasible for classifying the pageview stream in this case study.

5.2 The Raw Dataset

The raw data from that portal website is a dataset of web access logs, in which each record is a pageview. Each pageview can be viewed as a collection of web objects or resources representing a specific "user event", e.g., reading an article, viewing a product page, or adding a product to the shopping cart. For a concrete example, one pageview from that log dataset is essentially a string, which is composed of multiple fields such as time, IP address, and visited page.

```
2012-10-31 21:30:43     1.1.1.1     Mozilla/5.0+(compatible;+MSIE+9.0;+Windows+NT
    +6.1;+Win64;+x64;+Trident/5.0;+yie9)        http://www.startpagina.nl/      c9c
    6484956a93a9e457b3b873637c642                       portals_communities::www_
    index                           www_index_familyshopper_test      1580    1600x
    900x24x24   470027   16      {‘‘jv’’:‘‘1’’,‘‘ul’’:‘‘nl’’,‘‘lng’’:‘‘nl’’,‘‘idp
    ’’:‘‘2230412637174’’,‘‘re’’:‘‘1583x708’’,‘‘hl’’:‘‘22x30x41’’,‘‘hm’’:‘‘1’’,‘‘
    vtag’’:‘‘41003’’,‘‘s’’:‘‘470027’’,‘‘cn’’:‘‘lan’’,‘‘Language’’:‘‘nl-NL’’,‘‘ati
    ’’:‘‘PUB-[26305]-0’’}
```

To get a flavor of pageviews, we project seven pageviews into five straightforwardly meaningful fields: time, cookie, IP, page, and user agent, and these projected pageviews are listed in table 5.1, where each row represents a pageview, and each column represents a field [1]. Take the first pageview in table 5.1 as an example, it shows that a user with ip address `1.1.1.1` clicked page `index` (the index page of that website) using browser Internet Explorer version 9.0 on Windows 6.1 at 21:30:43 on 31st October, 2012.

Table 5.1: Example pageviews

p_id	time	cookie	ip	page	user agent
1	2012-10-31 21:30:43	c9c64849	1.1.1.1	index	...MSIE+9.0;+Windows+NT+6.1...
2	2012-10-31 21:31:05	fb46fda3	2.2.2.2	index	...Firefox/16.0;+Windows+NT+5.1...
3	2012-10-31 21:32:43	c9c64849	1.1.1.1	weer	...MSIE+9.0;+Windows+NT+6.1...
4	2012-10-31 21:33:05	fb46fda3	2.2.2.2	weer	...Firefox/16.0;+Windows+NT+5.1...
5	2012-10-31 21:34:20	c9c64849	1.1.1.1	buienradar	...MSIE+9.0;+Windows+NT+6.1...
6	2012-10-31 22:21:20	c9c64849	1.1.1.1	weer	...MSIE+9.0;+Windows+NT+6.1...
7	2012-10-31 22:31:20	c9c64849	1.1.1.1	buienradar	...MSIE+9.0;+Windows+NT+6.1...

The historical raw dataset comprises 37 days of pageviews, from 2012-10-01 to 2012-10-22, and from 2012-10-31 to 2012-11-14 [2]. There are totally 178,986,000 (179 million) pageviews, 112.5 GB in size. Simply training a classifier on this large dataset becomes infeasible, due to the limit of main memory and the running time of some accurate classifiers. Even worse, this large amount of pageviews indicates a high velocity streaming. In particular, there are 3,977,467 (4 million) pageviews per day, and 46 pageviews per second on average.

5.3 Pre-processing

The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. The raw dataset cannot directly be used as classifiable data, due to missing values, invalid values, and the missing of labels. From a pageview in the raw dataset, we cannot tell the user's action. To see a user's click when he was visiting the index page, single pageviews cannot tell us such information; even though there is a field indicating the user clicked or not, we still do not know what the previous page.

To recognize a user's clicks in pageviews, one effective way is to infer from *sessions*, which is a sequence of pageviews by a single user during a single visit. Identifying such sequences of pageviews is called sessionization. In general, sessionization groups the pageviews of a user in a

[1] The values of these fields are not exactly projected; meaningless characters are eliminated for clarity. And, column p_id is not present in raw data.

[2] The data of 2012-10-23 to 2012-10-30 is missing.

Table 5.2: Identified users

user	cookie	ip	user agent	p_id	time	page
User 1	c9c64849	1.1.1.1	...MSIE+9.0; +Windows+NT+6.1...	1	2012-10-31 21:30:43	index
				3	2012-10-31 21:32:43	weer
				5	2012-10-31 21:34:20	buienradar
				6	2012-10-31 22:21:20	weer
				7	2012-10-31 22:31:20	buienradar
User 2	fb46fda3	2.2.2.2	...Firefox/16.0; +Windows+NT+5.1...	2	2012-10-31 21:31:05	index
				4	2012-10-31 21:33:05	weer

certain time period. If there is a pageview which is not the index page in a session, we can safely assert that there is a user click. Knowing clicks in each session, we can label the first pageview in this session, and build predictive models on these pageviews.

5.3.1 Sessionization

Since a user may visit the website more than once, the server of this website may record multiple sessions for each user. Therefore, identifying sessions consists of two steps: identifying users, then identifying the sessions of each user.

User identification In an ideal case, each user is represented by a unique id in pageview logs, and users can easily be distinguished by their ids. However, in practice, most users are not registered, therefore there is no such field. In the absence of authentication mechanisms, the most widespread approach to distinguishing among unique visitors is the use of client-side cookies. But cookies are not sufficient, since client-side cookies are sometimes disabled by users due to privacy concerns. Another typical representations of users are IP addresses. However, IP addresses alone are not generally sufficient for mapping pageview entries onto the set of unique visitors. This is mainly due to the proliferation of ISP proxy servers which assign rotating IP addresses to clients as they browse the Web. Therefore, two occurrences of the same IP address (separated by a sufficient amount of time), in fact, might correspond to two different users [32]. Furthermore, multiple users may share a same external IP address if they are in a local area network.

Without user authentication or client-side cookies, it is still possible to accurately identify unique users through a combination of IP addresses and other information such as user agents and referrers [12]. In practice, we combine cookie, IP, and hence user agent to represent a user. Applying this strategy to identify users from the pageviews in Tab 5.1, there are two users: one from `1.1.1.1` having a cookie of `c9c64849` and using Internet Explorer 9.0 on Windows 6.1, and the other from `2.2.2.2` having a cookie of `fb46fda3` and using Firefox 16.0 on windows 5.1, as shown in Table 5.2.

Sessionization After identifying users with a combination of cookie, IP, and user agent fields, we are able to sessionize pageviews. Sessionization is the process of segmenting the pageviews of each user into sessions, each representing a single visit to the website. One simple but effective method for sessionization is a time-oriented heuristics. This heuristics assumes the time spent on a single page must not exceed a threshold δ, and separates sessions with this threshold as a time gap. The procedure of sessionization is described as follows. For a ordered sequence of pageviews by timestamp from an individual user, firstly assign the first pageview to a new constructed session. Then let t' be the timestamp of the pageview most recently assigned to a constructed session, the next pageview belongs to the same session if and only if its timestamp t'' holds that $t'' - t' \leq \delta$. Otherwise, this pageview becomes the first of the next constructed session.

In our sessionization, the threshold δ was set to 30 minutes. Applying this setting to sessionize the pageviews in Table 5.1 after user identification, the resulting sessions are shown in Table 5.3.

Table 5.3: Identified sessions

session	user	p_id	dttm	page
Session 1	User 1	1	2012-10-31 21:30:43	index
		3	2012-10-31 21:32:43	weer
		5	2012-10-31 21:34:20	buienradar
Session 2	User 2	2	2012-10-31 21:31:05	index
		4	2012-10-31 21:33:05	weer
Session 3	User 1	6	2012-10-31 22:21:20	weer
		7	2012-10-31 22:31:20	buienradar

We can clearly see there are three sessions, and two of them are belonging to User 1. Applying sessionization to the raw dataset, we obtain 75,663,545 sessions.

5.3.2 Labeling

Intuitively, user clicks can easily be detected in sessions —if the length of a session is more than 1, then there must be clicks. However, not all clicked pages are interesting, especially the page asking cookie acceptance. Therefore, we can safely judge there is at least one click unrelated to cookie in a session if the length of this session is larger than 1 and there is at least one page which is neither the index page nor related to cookie. Formally, for a session S,

$$\exists p \in S \colon p \notin \{\texttt{index}, \texttt{cookiepage}\} \Rightarrow \textit{click}$$

. That is, in session S, if there is at least one page which is not the index page and is not related to cookie, then S is labeled as *click*, otherwise non-click.

In predictive modeling, however, the labeled sessions are still not appropriate to be directly input into building classifiers. Because when a user visits the index page, we do not have sessions but only one pageview, though this pageview of the index page may lead to following pageviews. Therefore, utilizing the information in a session but not in the first pageview of this session is useless. Consequently, for each session, we only retain the first pageview, namely origin, to build our classification models.

Sometimes an origin is not on the index page of that website. As shown in Table 5.4 which represents sessions in Table 5.3 as paths, Session 3 starts from page `weer`, rather than `index`. Those sessions starting from pages other than `index` are removed during our pre-processing, since we only care about users visiting the index page. Fortunately, 64,555,924 (85.32%) sessions start from the index page.

There are 3,345,275 origins that led to clicks, and 61,210,649 led to non-clicks.

Table 5.4: Pageviews in a session as paths

session	pageviews (path)
Session 1	index→weer→buienradar
session 2	index→weer
session 3	weer→buienradar

5.3.3 Transformation

Raw fields in pageviews are likely to be not that suitable for pattern mining and classification. Meaningful information can be derived from raw fields. For instance, the number of pixels of a screen can be derived from a screen resolution in the representation of "1600×900". Furthermore, values of some fields may be aggregated by exisiting knowledge. For example, IP addresses can be

translated into cities, and time can be translated into different phases in a day such as morning and afternoon. Lastly, data granularities which are too fine would lead to overfitting.

Therefore, it is necessary to transform each raw field in pageviews into a more meaningful representation. Each raw field is transformed to one or more attributes which are of type string. And we use attribute-value pairs to represent the dataset as a binary one. The detailed explanations of fields and transformations are given in Appendix A. And a brief summary of raw fields and the numbers of the attributes transformed from them are shown in Table 5.5.

In summary, there are 19 raw fields in each pageviews, and then they are transformed into 25 attributes in total.

Table 5.5: Transformation of raw fields

Field	Description	#attributes
`dttm`	date and time	2
`ip`	IP address	1
`referer`	referer	1
`screen`	screen resolution	2
`site_section`	section of the page	1
`user_agent`	user agent	4
`vtag`	tracker's version	1
`jv`	java vm	1
`re`	browser's resolution	2
`lng`	browser language	1
`ati`	unknown	1
`hm`	if set homepage	1
`cn`	connection type	1
`anc`	unknown	1
`anct`	unknown	1
`vrn`	new visitor	1
`roinbh`	Amount of time that has gone by since purchase was made	1
`UA_PROF`	mobiles UA profile	1
`ref`	referer	1

5.4 The Offline Setting

Before applying our pattern-based classification approach to a real streaming environment, we validate this approach in an offline setting to discover a parameter setting of our approach that may best fit the data of this study case. In addition, we experimentally explore the effect of patterns in classification: to what extent patterns can help classification performance.

Our approach is involved in undersampling, sampling a given number of discriminative patterns, post-processing sampled patterns with a given dataset coverage threshold, and building classification models. Different parameters are involved in this approach. Specifically in this offline setting, we discover the size of the training set, the dataset coverage threshold in post-processing, and the size of sampled discriminative patterns. Besides, encoding instances with or without original attributes are considered as well. Additionally, we compare the classification performances of two typical classifiers: naive Bayes and decision trees (C4.5[38]) [3].

5.4.1 The Balanced Training Set

The dataset of this offline setting comprises the pre-processed data of two consecutive days, in which we build classification models over the first day, and validate the models on the second day. The pre-processed data of two consecutive days, 2012-10-01 and 2012-10-02, are original pageviews

[3]The implementations in Weka [24] with default parameters were used in our experiments.

of sessions. Less than 5% of these pageviews led to clicks, as shown in Table 5.6, which reports the class distribution of these pageviews.

Table 5.6: Class distribution of origins

Dataset	Positive	Negative	Total
2012-10-01	89061(4.32%)	1972913(95.68%)	2061974(100%)
2012-10-02	92273(4.66%)	1886456(95.34%)	1978729(100%)

To build a balanced training set from the data of 2012-10-01, we undersampled from both the positives and the negatives. The undersample size was empirically set to 60000, i.e. in the undersampled balanced training set, there were 30000 positives and 30000 negatives respectively.

To assess the performance of a classifier on skewed datasets, accuracy is no longer a good overall measure since the majority class dominates the result. We introduce the area under ROC curve (AUC) as the evaluation metric. The ROC curve shows the trade-off between true positive rate and false positive rate for a given classifier. A good classifier would produce a ROC curve as close to the left-top corner as possible. The area under the ROC curve (AUC) is a measure of the model accuracy, in the range of $[0, 1]$. A perfect model will have an area of 1.

Table 5.7 shows our undersampling technique can significantly improve the recall of the positives in the test set. Two kinds of classifiers, naive Bayes and decision trees, trained with or without undersampling from the data of 2012-10-01 were tested on the data of 2012-10-02. AUC, precision and recall of each class are reported. With the naive Bayes classifier trained on the undersampled balanced training set (denoted as "nb-undersample"), the recall of the positive class drastically increases from 0.030 to 0.637 comparing with another naive Bayes classifier trained on the whole data of 2012-10-01. However, the overall measure, AUC, slightly drops with the undersample; nevertheless, significantly improving the recall of the positives is worth sacrificing a minor AUC. For the decision trees, the recall of the positive class are improved from 0.000 to 0.748 with undersampling. Both the precision and the recall of the positives equal to 0 indicates that the decision tree without undersampling classified all instances as negative. In this case, decision trees seem useless without undersampling.

Table 5.7: Classification performance with undersampling

Classifier	AUC	0-precision	0-recall	1-precision	1-recall
nb-undersample	0.696	0.974	0.656	0.083	**0.637**
nb-no-undersample	0.700	0.954	0.990	0.129	0.030
tree-undersample	0.680	0.978	0.556	0.076	**0.748**
tree-no-undersample	0.500	0.953	1.000	0.000	0.000

In brief, the *original* training set used in our offline experiments is a random undersample consisting of 30000 positives and 30000 negatives from the pageviews of 2012-10-01; the test set is all the pageviews of 2012-10-02.

The baseline model used in later offline experiments is a naive Bayes classifier trained on this original training set, with a 0.696 AUC as its classification performance.

5.4.2 Pattern-based Classification

In our pattern-based classification approach, we mine positive-discriminative patterns from the balanced training set, then post-process them before encoding instances. The size of a post-processed pattern collection depends on the dataset threshold δ. Intuitively, larger δ would lead to more patterns, while smaller δ would lead to less. The number of patterns that used in classification may affect the classification performance of different encoding strategies.

Apart from the dataset coverage threshold, another variable is the size of patterns that should be sampled. Due to the sampling essence, increasing the size of a pattern sample may have a

better chance to obtain more patterns with higher discriminativities; however, larger sample sizes need more computing resources.

In the following, the influences of the dataset coverage threshold and the pattern sample size are experimentally explored.

Dataset Coverage Threshold

To explore the influence of the dataset coverage threshold on patterns, we mined a fixed number of patterns, i.e. 1000, from the balanced training set, then post-processed them with dataset coverage thresholds varying from 1 to 4, leading to 4 collections of patterns.

In Table 5.8, statistics of the post-processed pattern collections are reported. The dataset coverage threshold, or just coverage in short, means the times that the selected patterns can cover the dataset. Information gain (IG) is the discriminative measure. A larger information gain of a pattern indicates the dataset is more separable by that pattern. As can be seen, more coverages indeed lead to more selected patterns, as well as longer pattern lengths. On the other hand, average frequency and information gain drop with increasing coverage, since latter selected patterns would have less discriminative power due to the limited number of instances they fit.

Table 5.8: Statistics of post-processed patterns

Coverage	Number	Median length	Median frequency	Median IG
1	6	2	0.54	0.0562
2	17	3	0.45	0.0536
3	28	3	0.45	0.0448
4	70	5	0.17	0.0055

The usefulness of these 4 collections of post-processed patterns are explored by integrating them into classification modeling. In particular, for each pattern collection, we encoded all instances *1)* only with the patterns in the collection, and *2)* with with the patterns in the collection and original attributes of instances.

For both strategies, we trained a naive Bayes classifier and a decision tree classifier, leading to four classification methods, *1) nb-pattern-only*: a naive Bayes classifier trained on the training set only with patterns as features; *2) nb-augmented*: a naive Bayes classifier trained on the training set with both original attributes and patterns as features; *3) tree-pattern-only*: a decision tree classifier trained on the training set only with patterns as features; and *4) tree-augmented*: a decision tree classifier trained on the training set with both original attributes and patterns as features. Then each classification model was tested on a correspondingly encoded test set, which is based on the whole data of 2012-10-02.

Figure 5.1 shows the AUCs of different classification methods with coverages ranging from 1 to 4. We can see that nb-augmented consistently performs the best in terms of AUC, with all values of coverage; though it slightly drops as coverage increases. Both classifiers on augmented training sets perform better with a smaller coverage; and perform worse using the pattern-only encoding strategy with larger coverages. This may be because the patterns selected with $\delta = 4$ are still too general in both the positives and negatives. One may notice that only nb-augmented with $\delta = 1$ outperforms the baseline—the naive Bayes classifier built with original attributes of the balanced training set. So in the sense of AUC, only the naive Bayes classifier helps classification performance.

In brief, the "pattern-only" encoding strategy performs better with $\delta = 4$; and the "augmented" strategy performs better with $\delta = 1$.

Pattern Sample Size

Another factor that may impact on the usefulness of selected patterns is the sample size of discriminative patterns. More sampled patterns contribute to higher possibilities to include discriminative patterns, but require more computational cost.

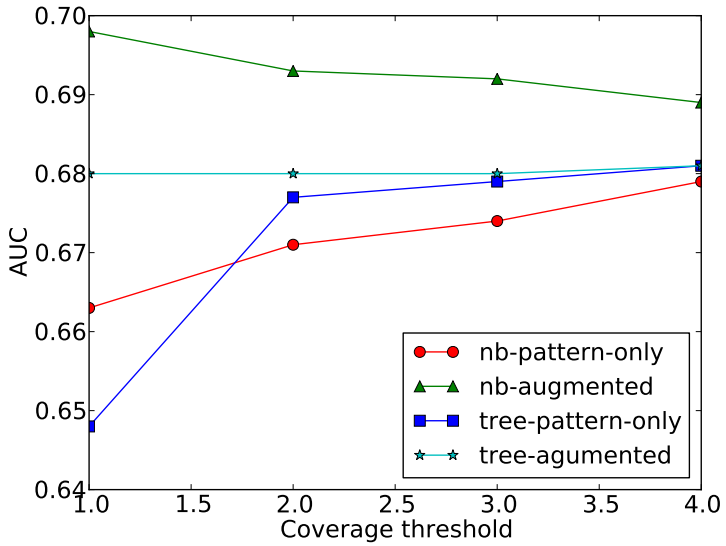


Figure 5.1: AUC with different dataset thresholds

To explore the influence of the size of pattern samples, we mined different numbers of discriminative patterns from the balanced training set. The range of numbers was empirically set from 1000 to 10000, with an interval of 1000. According to the experiments on varying the dataset coverage threshold, we post-processed each mined pattern collection with coverage δ equal to 1 and 4, facilitating the "augmented" and "pattern-only" encoding strategies respectively.

The statistics of the post-processed patterns are reported in Table 5.9. As can be seen, the qualities of the patterns with both coverages do not vary with the size of mined pattern samples, since the median frequencies and median information gains do not vary much. Though the numbers of the post-processed patterns with both coverages change with the sample size, this change hardly reflects any obvious relationship between them. Therefore, post-processed patterns from a 1000 sized sample may have the same quality as those from samples of other sizes.

Table 5.9: Statistics for the post-processed patterns

	Coverage 1				Coverage 4			
Size	No.	Median len.	Median freq.	Median IG	No.	Median len.	Median freq.	Median IG
1000	6	2	0.54	0.0562	70	5	0.17	0.0055
2000	8	3	0.53	0.0562	49	4	0.43	0.0488
3000	4	3	0.49	0.0562	24	3	0.53	0.0557
4000	4	2	0.48	0.0562	24	3	0.51	0.0562
5000	5	3	0.51	0.0562	31	3	0.50	0.0556
6000	4	1.5	0.48	0.0562	28	3	0.53	0.0562
7000	7	3	0.52	0.0562	42	3.5	0.52	0.0547
8000	5	3	0.52	0.0562	41	3	0.45	0.0545
9000	6	2	0.53	0.0562	27	2	0.45	0.0557
10000	8	2.5	0.53	0.0562	32	3	0.52	0.0559
stdev	1.57	0.58	0.02	0.0000	14.35	0.79	0.11	0.0157

To further explore the pattern qualities in classification, we encoded instances using the "pattern-only" strategy with the patterns post-processed with coverage equal to 4, and encoded instances using the "augmented" strategy with the patterns post-processed with coverage equal to 1. Both the naive Bayes classifier and decision tree classifier are used with these two encoding strategies, leading to four classification methods: 1) *nb-pattern-only*: a naive Bayes classifier

trained on the training set only with patterns as features, where the patterns are post-processed with $\delta = 4$; 2) *nb-augmented*: a naive Bayes classifier trained on the training set with both original attributes and patterns as features, where the patterns are post-processed with $\delta = 1$; 3) *tree-pattern-only*: a decision tree classifier trained on the training set only with patterns as features, where the patterns are post-processed with $\delta = 4$; and 4) *tree-augmented*: a decision tree classifier trained on the training set with both original attributes and patterns as features, where the patterns are post-processed with $\delta = 4$.

Figure 5.2 shows the AUCs of these classification methods varying the number of sampled patterns. As can be seen, the size of pattern samples has little influence on the AUCs of the classifiers with the "augmented" encoding strategy. Whereas, this size flunctuates the AUCs of the classifiers with the "pattern-only" encoding strategy. Besides, nb-augmented consistently outperforms the others, and it is the only one that has a better performance than the baseline, which is a naive Bayes classifier trained on the original training set.

Therefore, the size of a pattern sample is unlikely to affect the classification performance of the corresponding post-processed patterns. 1000 is a emperically satisfactory size of pattern samples in our case.

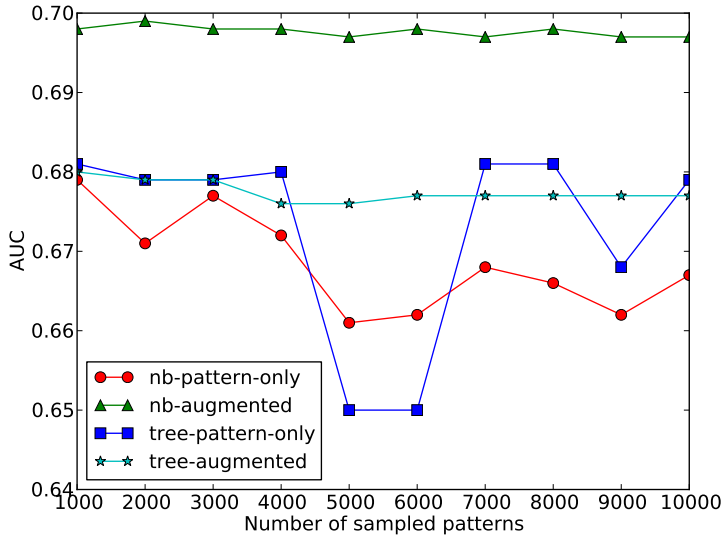


Figure 5.2: AUC with different sample sizes of patterns

Detailed Influence of Patterns on Classification

We then investigate the detailed performances of classification methods described above with the best setting. That is, the number of sampled patterns is 1000; the "pattern-only" strategy is used with $\delta = 4$; the "augmented" strategy is used with $\delta = 1$. We trained *1)* a naive Bayes classifier "nb-pattern-only" with the "pattern-only" encoding strategy; *2)* a naive Bayes classifier "nb-augmented" with the "augmented" encoding strategy; *3)* a decision classifier "tree-pattern-only" with the "pattern-only" encoding strategy; *4)* a decision classifier "tree-augmented" with the "augmented" encoding strategy. Notice that these four classification models are identical to the four in the last experiment with the sample size equal to 1000.

Table 5.10 shows the detailed classification performance of the classifiers described above plus two baselines: a naive Bayes classifier and a decision tree classifier trained without patterns, called "nb-original" and "tree-original" respectively. As can be seen, nb-augmented achieves the overall performance in terms of AUC, which surpasses all the decision trees, but is merely slightly better than "nb-original". Comparing with "nb-original", "nb-augmented" trades the recall of the

positives for the precision of the positives. In terms of the recall of the positives, the decision trees have better performance than the naive Bayes classifiers, possibly due to the rule-based property of decision trees. Augmenting instances with patterns can slightly improve the recall of the positives for both the naive Bayes classifiers and decision tree classifiers.

Table 5.10: Detailed metrics of different classifiers

Classifier	AUC	0-precision	0-recall	1-precision	1-recall
nb-original	0.696	0.974	0.655	0.083	0.638
nb-pattern-only	0.679	0.972	**0.684**	**0.085**	0.602
nb-augmented	**0.698**	0.972	0.681	**0.085**	0.606
tree-original	0.680	0.978	0.556	0.076	0.748
tree-pattern-only	0.681	0.978	0.571	0.077	0.723
tree-augmented	0.680	**0.979**	0.548	0.076	**0.756**

Figure 5.3a plots the ROC curve of the naive Bayes classification models. The curves of nb-original and nb-augmented are nearly overlapped, indicating that the classification performance cannot be much improved by patterns. It conforms to the results in Table 5.10: nb-augmented improves AUC by 0.002 comparing with nb-original.

We further show the precision-recall curve of the naive Bayes classifiers in Figure 5.3b. When the recall is less than 0.6, the model trained on the augmented training set performs better than the models trained on only patterns or original attributes. While, when the recall is greater than 0.6, the difference of classification performance is not obvious.

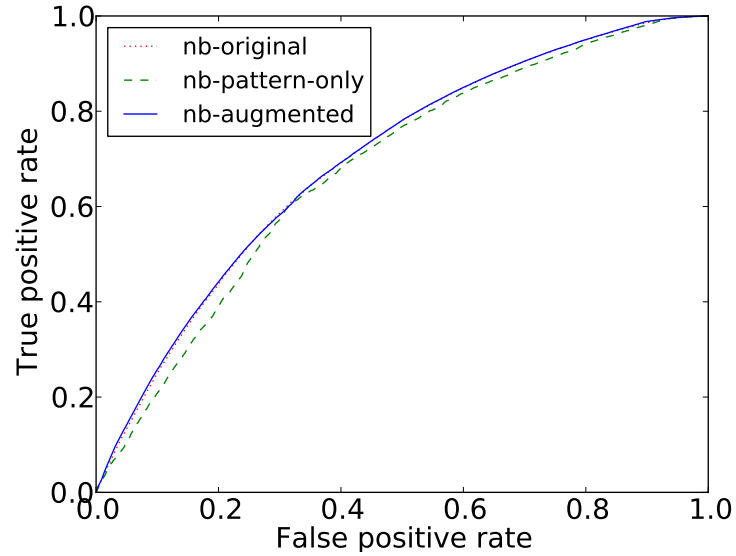
5.4.3 Conclusion

To summarize, we can answer the effects of patterns on classification, *1)* patterns can increase the recall of the negatives but decrease the recall of the positives for naive Bayes classifiers; *2)* a slightly better AUC can be achieved by using patterns in naive Bayes classifiers.

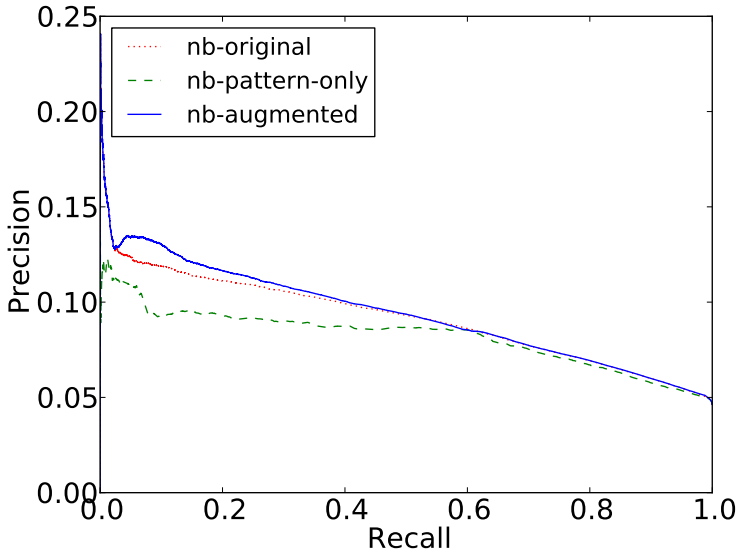
The coverage threshold and the size of a pattern sample do not affect the classification performance of naive Bayes classifiers and decision tree classifiers with the "augmented" encoding strategy, but affect that with the "pattern-only" strategy. The "pattern-only" encoding strategy can hardly outperform the "augmented" one.

Finally, the empirical parameters that we have found are:

1. the undersample size is set to 60000, with 30000 positives and 30000 negatives respectively;
2. the sample size is set to 1000, followed by a post-processing with the dataset coverage threshold equal to 1;
3. a naive Bayes classifier with the "augmented" encoding strategy is favored.



(a) ROC curve of naive Bayes



(b) Precision-recall curve of naive Bayes

Figure 5.3: Classification performance with patterns

5.5 The Simulating Online Setting

To see whether our patterns-based classification approach is applicable to the click prediction problem, ideally we should deploy it into a real streaming environment. However, since the real-world data we have is stored in a static dataset instead of being generated as a data stream, we cannot really apply our approach to a real streaming setting. Nevertheless, we can simulate a data stream from the static dataset.

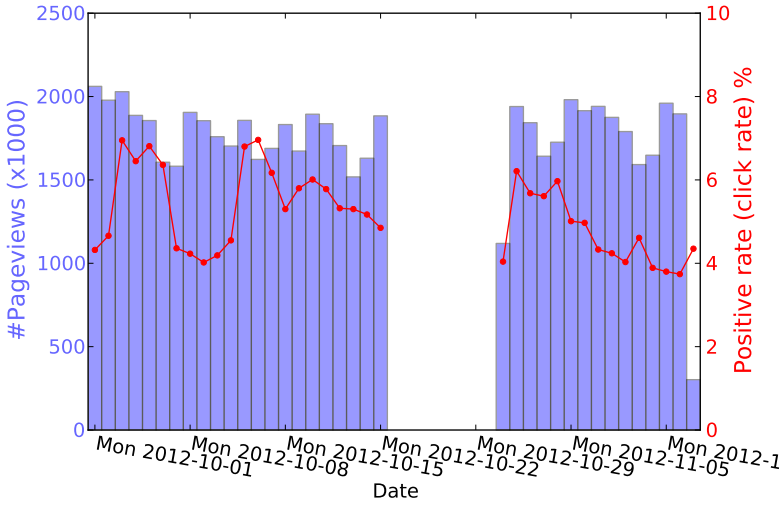
In this section, we will describe the application of the approach described in Chapter 4, and compare different classification performances of different sizes of windows. Then we will answer whether our approach is applicable for the click prediction of this specific study case.

5.5.1 Setup

The Dataset

The simulating data stream comes from the pre-processed pageviews of 37 days, from 2012-10-01 to 2012-11-14 with 8 days missing. The numbers of the original pageviews, or origins, in each day and the click rates are shown in Figure 5.4. The bars represent the numbers of original pageviews, with 2012-10-23 to 2012-10-30 missing. The polygonal curve represents the click rates.

Figure 5.4: Dataset statistics



Sliding Window

As described in Chapter 4, we adopt a sliding window to handle the data stream. The variables in the sliding window consists of the window size w days, and the window step d days. In our experiments, we fix the window step to 1 day. That is, the window slides when after one day from the time point it lastly slides. We will test the classification performances with different window size w.

After the window slides, we apply our pattern-based approach within this window to construct classification models for predicting incoming pageviews. The setting follows the result of the offline experiments described in last section. Specifically, we undersampled a balanced training set from the pageviews within this window, then employed the pattern mining, post-processing and model building as described in Section 5.4. The undersampled balanced training set consisted of 30000 positives and 30000 negatives respectively; the size of pattern samples was set to 1000; and the dataset coverage threshold in post-processing was set to 1 and 4 (for different encoding strategies respectively). Since the classification performance of decision trees hardly exceeded that of naive Bayes classifiers in the offline experiments, decision trees were not considered in the simulating streaming experiments.

Therefore there are two strategies to be tested: 2) *wd-original*, naive Bayes classifiers that trained within a sliding window with original attributes; 3) *wd-augmented*, naive Bayes classifiers that trained within a sliding window with original attributes and patterns post-processed with cover 1.

The window size w in our experiments is set to 1 and 5. When we use a 1-day sized window, it is essentially batched processing: train a model from the data in a day, then apply this model to

the next day. For the 5-day sized window, when the number of days are not sufficient to form the 5-day sized window, all the days are included in this window. For example, when a 5-day sized window just starts, it includes the day 2012-10-01 only. Furthermore, the sliding window slides over the missing days, As a result, the sliding window slided 36 times, and 36 days, 2012-10-02 to 2012-10-22 and 2012-10-31 to 2012-11-14, were tested on.

The Baseline

We also employed a naive Bayes classifier trained on an undersample of the first day without any pattern to predict the days from 2012-10-02 to 2012-11-14 as a baseline: *nw-original*, a naive Bayes classifier trained on an undersample of the first day with original attributes without a sliding window.

The weighted F-measure described in Chapter 2 is used as the evaluation metric.

5.5.2 Results

We first report the classification performance of our pattern-based approach comparing with the baseline, then report the comparison of 1-day sized window and 5-day sized window in terms of classification performance.

Comparisons with The Baseline

Figure 5.5 shows the classification performance results of three classification methods in a 1-day sized sliding window on these 36 days. As can be seen, 1d-augmented consistently performs better than the baseline and 1d-original, except at the end of the time period, in which 1d-original shows a better performance. It is somewhat surprising that 1d-original is not obviously better than the baseline; or even worse, it is often surpassed by the baseline. Thus, 1d-augmented is likely to slightly improve the classification performance upon the baseline in terms of the weighted F-measure, but 1d-original not.

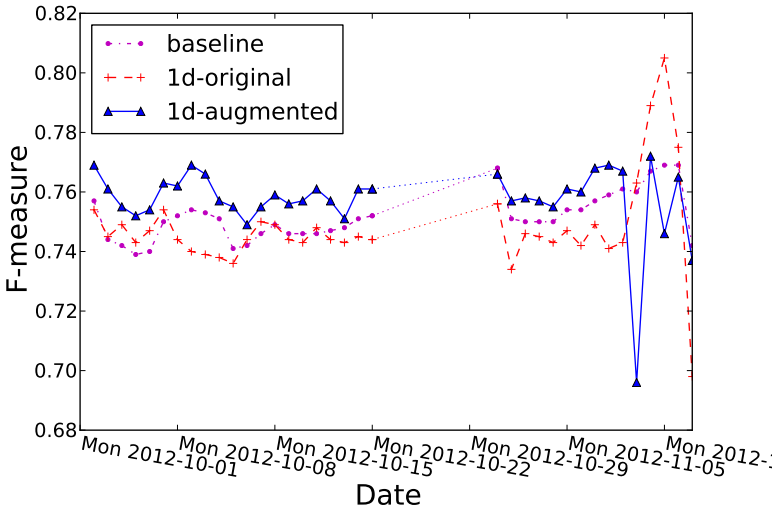


Figure 5.5: Weighted F-measure of the 1-day sized sliding window

Figure 5.6 shows the weighted F-measures of the 5-day sized sliding window. The same trend as Figure 5.5 can be observed. The 5-day sized window with patterns outperform the baseline and

5d-original. Again surprisingly, the baseline sometimes performs better than 5d-original.

In brief, both 1-day and 5-day sized windows can achieve better performances than the baseline. In addition, there is unlikely to be concept drift in our data, as there is no strong evidence that the classification performance with a sliding window is consistently better than that without a sliding window.

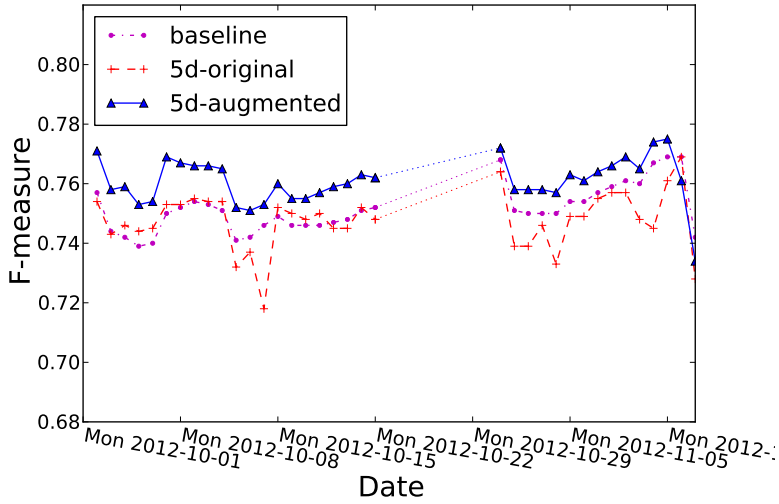


Figure 5.6: F-measure of the 5-day sized sliding window

Comparison of Windows

Figure 5.7 shows the comparison of the 1-day and 5-day sized windows. Similarly to the results we have shown, they both have a better performance than the baseline. However, the difference between these two windows is very subtle. One may notice that at the tail, 5d-augmented still has a stable performance, but 1-day does not. This may be because some patterns are captured by 1d-augmented, but those patterns are only existing in a very short period, namely, one day. Therefore, these patterns cannot generalize in the next day, and utilizing them in classification tends to be unhelpful, or even worse.

Overall Comparison

We report the AUCs and the weighted F-measures on average of each method to give an overall comparison. As can be seen, the AUCs of all the methods are nearly the same. The classification performances are only distinguished by F-measure. Specifically, 5d-augmented achieves the best performance, and surpasses the baseline by 0.009 on average.

Table 5.11: Average AUC and weighted F-measure

Method	AUC	F-measure
baseline	0.673 ± 0.020	0.752 ± 0.008
1d-original	0.674 ± 0.022	0.748 ± 0.016
1d-augmented	0.675 ± 0.022	0.757 ± 0.013
5d-original	0.675 ± 0.021	0.748 ± 0.010
5d-augmented	0.675 ± 0.022	$\mathbf{0.761 \pm 0.008}$

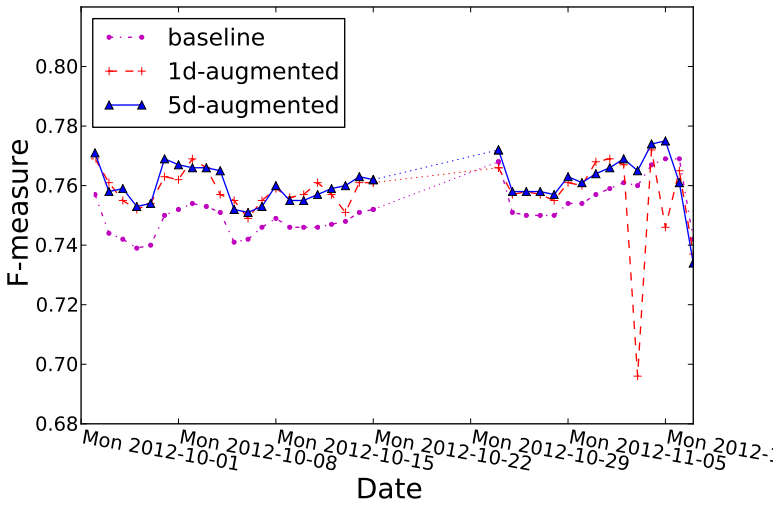


Figure 5.7: F-measure the sliding windows

5.5.3 Conclusion

The experiments show that our pattern-based classification approach can slightly improve the classification performance upon the baseline by 0.009 in terms of the average weihghted F-measure. and by 0.002 in terms of the average AUC. Due to the indication from the offline experiments, the contributions to the improvement of the average weighted F-measure may be that, at a same recall of the positives, pattern-based classification can achieve a higher precision.

Additionally, since the baseline performs almost the same as a classifier trained on original attributes within a sliding window, it might be not necessary to introduce sophisticated methods to handle concept drift.

5.6 Summary

To explore the usefulness of our pattern-based approach in a real-world environment, we conducted experiments with a case study. The raw data of this case study are pageviews, and were preprocessed to be classifiable instances.

We experimentally selected parameters for our approach with experiments on a small proportion of the whole dataset, then applied our approach with the selected parameters using a sliding window to handle a simulating data stream. In experiments, our approach slightly surpasses a classifier statically trained on the initial day with original attributes.

Chapter 6

Conclusions

In this thesis, we have tackled the click prediction problem by the pattern-based classification approach. The critical problems in click prediction are weak features, skewed class distribution and streaming data, which are addressed by grouping individual features, random undersampling, and a sliding window technique respectively.

For a side problem raised by the pattern-based classification approach, mining patterns from large datasets has been addressed by a distributed pattern sampling algorithm proposed by us. The theoretical analysis show that the memory requirements of the proposed algorithm only dependent on the number of desired patterns. The experimental evaluation shows the scalability of the algorithm, that is, the computation time is roughly linear dependent on the number of patterns to sample, which makes the algorithm suitable in most practical settings.

To handle the skewed data stream with the pattern-based classification approach, we adopt a sliding window to capture the high-velocity stream, and undersample a balanced dataset as the training set within a window for both pattern mining and model learning.

We have validate our pattern-based approach with a case study concerned with a Dutch portal website, whose purpose is to predict user clicks on their index webpages. The experiments conducted in an offline setting show our pattern-based classification performs slightly better than classification only with original features in terms of AUC.

In a simulating online setting, the experiments show our pattern-based approach can achieve an average AUC of 0.675 over a period of 36 days, and surpass the classification model without pattern by 0.002. Besides, the average weighted F-measure of our approach is 0.009 higher than the baseline. These results justify that our approach can indeed improve classification performance.

Finally, from the industrial view, the question that whether our approach should be deployed in a real streaming setting leaves to the end user: is a 0.002 improvement in AUC and a 0.009 improvement in weighted F-measure worth deployment?

Bibliography

[1] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer, 2007. 24

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993. 6

[3] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994. 7

[4] S. D Bay and M. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001. 7

[5] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 99:1601–1604, 2010. 23

[6] M. Boley, S. Moens, and T. Gärtner. Linear space direct pattern sampling using coupling from the past. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012. 22

[7] Mario Boley, Claudio Lucchese, Daniel Paurat, and Thomas Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 582–590. ACM, 2011. 7, 11, 12, 13, 14, 22

[8] Joong Hyuk Chang and Won Suk Lee. Finding recent frequent itemsets adaptively over online data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 487–492. ACM, 2003. 25, 30

[9] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004. 23, 24

[10] Haibin Cheng and Erick Cantú-Paz. Personalized click prediction in sponsored search. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 351–360. ACM, 2010. 1, 2

[11] Hong Cheng. *Towards Accurate and Efficient Classification: A Discriminative and Frequent Pattern-Based Approach*. ProQuest, 2008. 1, 32

[12] Robert Cooley, Bamshad Mobasher, Jaideep Srivastava, et al. Data preparation for mining world wide web browsing patterns. *Knowledge and information systems*, 1(1):5–32, 1999. 35

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 5

[14] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 11

[15] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000. 31

[16] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 43–52, 1999. 7

[17] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 43–52. ACM, 1999. 6

[18] P. Efraimidis and P. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181 – 185, 2006. 14, 15

[19] M. El-Hajj and O. Zaiane. Parallel leap: large-scale maximal pattern mining in a distributed environment. In *IEE International Conference on Parallel and Distributed Systems*, 2006. 22

[20] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Citeseer, 2001. 24

[21] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005. 24, 25

[22] Jing Gao, Wei Fan, Jiawei Han, and S Yu Philip. A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, 2007. 30

[23] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Discovery science*, pages 278–289. Springer, 2004. 7

[24] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. 37

[25] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings ACM SIGMOD*, pages 1–12. ACM, 2000. 7, 22

[26] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994. 5

[27] Nathalie Japkowicz. Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, 42(1-2):97–122, 2001. 24

[28] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186, 1997. 24

[29] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang. Pfp: parallel fp-growth for query recommendation. In *ACM RecSys*, 2008. 17, 20, 22

[30] Wenmin Li, Jiawei Han, and Jian Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369–376. IEEE, 2001. 28

[31] Charles X Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *KDD*, volume 98, pages 73–79, 1998. 24

[32] Bing Liu. *Web data mining: exploring hyperlinks, contents, and usage data*. Springer, 2007. 35

[33] Marcus A Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *ICML-2003 workshop on learning from imbalanced data sets II*, volume 2, 2003. 24

[34] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *The Journal of Machine Learning Research*, 2:139–154, 2002. 24

[35] Mohammad M Masud, Qing Chen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Classification and novel class detection of data streams in a dynamic feature space. In *Machine Learning and Knowledge Discovery in Databases*, pages 337–352. Springer, 2010. 31

[36] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998. 19

[37] Foster Provost. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI2000 workshop on imbalanced data sets*, 2000. 24

[38] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993. 5, 28, 37

[39] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001. 24

[40] Pang-Ning Tan et al. *Introduction to data mining.* Pearson Education India, 2007. 5

[41] David MJ Tax and Robert PW Duin. Support vector domain description. *Pattern recognition letters*, 20(11):1191–1199, 1999. 24

[42] Chieh-Jen Wang and Hsin-Hsi Chen. Learning user behaviors for advertisements click prediction. In *Proceedings of the 34rd international ACM SIGIR conference on research and development in information retrieval Workshop on Internet Advertising*, pages 1–6, 2011. 1, 2

[43] Taifeng Wang, Jiang Bian, Shusen Liu, Yuyu Zhang, and Tie-Yan Liu. Psychological advertising: exploring user psychology for click prediction in sponsored search. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '13, pages 563–571, New York, NY, USA, 2013. ACM. 1

[44] Gary M Weiss and Foster J Provost. Learning when training data are costly: the effect of class distribution on tree induction. *J. Artif. Intell. Res.(JAIR)*, 19:315–354, 2003. 24, 30

[45] M. Zaki. Parallel and distributed association mining: A survey. *Concurrency, IEEE*, 7(4), 1999. 22

Appendix A

Transformation of raw fields

Here is described the transformation the raw fields in original pageviews of sessions to classifiable attributes. Each pageview was generated from an Internet tracker, as called Tracker here. Thus, the fields in each pageview can be divided into two parts: the fields appended by Tracker, and the fields (parameters) from the raw query string. There are 19 raw fields in each pageview, and we transform them into 25 attributes, which are all of type categorized.

Table A.1 gives the information about the transformation from fields to attributes. The name, the description and an example value of a field are given in each row, plus the attributes after transformation and the method of the transformation.

Table A.1: Transformation of fields

Field	Description	Example value	Derived attributes	Transformation
dttm	visiting time	2012-10-01 20:30:53	holiday, day phase	wether the time is holiday; the phase in the day, e.g. morning or evening
ip	visitor's IP address	1.1.1.1xx	city	the city of this IP address
referer	referer from Tracker	''	referer	translate into netloc/path/querystring
screen	screen resolution	1600x900x32x32	ratio, #pixel	discritize the ratio into 4 bins; discritize the number of the pixels into 6 bins
site_section	section of the page	16	section	map size_section into plain text according to a mapping file
user_agent	user agent	''	distribution, os, browser, mobile	retrive the device distribution, os name and version, browser name and version, and wether the device is mobile
vtag	Tracker version	''	vtag	keep
jv	java vm	1	jv	keep
re	browser's resolution	1500x800	ratio, pixel	as screen
lng	browser language	en-us	language	only keep the first part
ati	unknown	-	ati	keep
hm	if set homepage	1	hm	keep
cn	connection type	lan	cn	keep
anc	unknown	''	anc	keep
anct	unknown	''	anct	keep
vrn	new visitor	1	vrn	keep
roinbh	amount of time	23	roinbh	discritize in to 6 bins
UA_PROF	mobile's UA profile	''	ua	keep
ref	referer	''	ref	translate into netloc/path/querystring