De l'importance du pré-traitement des données pour l'utilisation de l'inférence grammaticale en *Web Usage Mining*

Thierry Murgue

Eurise – Université Jean Monnet
23 rue du docteur Paul Michelon
42023 Saint-Étienne Cedex 2
thierry.murgue@univ-st-etienne.fr

Résumé. Le *Web Usage Mining* est un processus d'extraction de connaissance qui permet la détection d'un type de comportement usager sur un site internet. Cette tâche relève de l'extraction de connaissances à partir de données : plusieurs étapes sont nécessaires à la réalisation du processus complet. Les données brutes, utilisées et souvent incomplètes correspondent aux requêtes enregistrées par un serveur. Le pré-traitement nécessaire de ses données brutes pour les rendre exploitables se situe en amont du processus et est donc très important. Nous voulons travailler sur des modèles structurés, issus de l'inférence grammaticale. Nous détaillons un ensemble de techniques de traitement des données brutes et l'évaluons sur des données artificielles. Nous proposons, enfin, des expérimentations mettant en évidence l'affectation des algorithmes classiques d'inférence grammaticale par la mauvaise qualité des logs bruts.

1 Introduction

Le *Web Usage Mining* a été introduit pour la première fois en 1997 (Cooley et al. 1997). Dans cet environnement, la tâche est d'extraire de manière automatique la façon dont les utilisateurs naviguent sur un site web. Depuis 1995, Catledge et Pitkow ont étudié la manière de catégoriser les comportements utilisateurs sur un site web (Catledge 1995). Le processus d'extraction de connaissance – pré-traitement, fouille, interprétation – est basé sur la disponibilité de données fiables : divers travaux on été menés sur la façon de traiter les données récupérables depuis un site web (Cooley et al. 1999, Pitkow 1997, Chevalier et al. 2003). Une grande majorité de chercheurs utilisent de manière systématique les informations contenues dans les enregistrements du serveur (fichiers de logs), mais ces données, sous forme brute, ne sont pas complètes : un pré-traitement est donc nécessaire. L'étape suivante du Web Usage Mining consiste à apprendre des modèles de comportement utilisateurs depuis ces données. Ainsi, ces dernières années, de nombreuses méthodes de traitement (Tanassa et al. 2004) et d'apprentissage ont été utilisées dans ce domaine : recherche de séquences fréquentes (Frias-Martinez 2002, Gery 2003), travaux sur l'utilisation de modèles structurés de type chaîne de Markov ou modèle de Markov caché (Hmm) (Pitkow 1999, Bidel et al. 2003). Certains chercheurs ont notamment travaillés sur des modèles grammaticaux : certains (Borges 1999) en utilisant des *n-grams*, d'autres (Karampatziakis et al. 2004) en étudiant le comportement

d'une méthode classique d'inférence grammaticale sur des automates probabilistes.

Dans cet article, nous présentons tout d'abord les deux types de problèmes liés à l'architecture des réseaux pouvant provoquer une corruption des données. Nous présentons dans la section 3, nos méthodes de reconstruction des logs, en donnant des résultats montrant la capacité de notre méthode à améliorer les données initiales. Ces expérimentations sont menées sur des données générées artificiellement pour posséder un échantillon de logs de référence. Nous présentons, ensuite, nos travaux actuels concernant un protocole d'expérimentations ayant pour but de faire émerger la réelle nécessité de la reconstruction des logs pour l'utilisation de l'inférence grammaticale en comparant cette méthodes d'apprentissage à de l'extraction de séquences fréquentes. Enfin nous concluons sur les travaux restant à mener.

2 Problèmes liés aux fichiers de logs

La quasi-totalité des travaux sur le Web Usage Mining utilisent les fichiers de logs enregistrés par le serveur de manière chronologique. Le protocole `Http`, utilisé dans les navigations web, est dissymétrique : en règle générale, un client demande une page que lui renvoie ou non (page inexistante, droits insuffisants, ...) le serveur. À chaque réponse le serveur enregistre un log fréquemment composé (Luotonen 1999) de l'adresse de la machine d'où provient la demande (adresse du client), un champ (souvent vide) concernant l'identification de l'utilisateur sur la machine cliente, un champ d'authentification dans le cas de requête « sécurisée », la date de la requête, la page demandée (la requête elle-même), le type de réponse du serveur (réussi ou non), la taille de la réponse. On trouve souvent deux champs supplémentaires : la page référente (l'`Url` de la page vue par l'utilisateur, lorsqu'il fait la requête), et le type de client utilisé. Dans un cas idéal, où chaque utilisateur utilise une machine différente et où toutes les machines sont reliées directement au serveur, ce type de données permet d'extraire de manière linéaire des *sessions ou visites utilisateur* [1]. Lorsqu'un utilisateur navigue sur un site dans un but précis, il effectue un ensemble de requêtes que l'on appellera *visite* ; s'il navigue à nouveau sur le même site dans un autre but, une nouvelle visite doit être crée. Il suffit, dans une première approximation, de regrouper toutes les requêtes d'une même provenance dans une fenêtre temporelle déterminée pour créer la visite.

L'architecture hiérarchique des réseaux et les facultés de *cache* et *proxy* rendent la tâche beaucoup plus difficile.

Dans (Pitkow, 97), la *mise en cache* est définie comme un mécanisme visant à restreindre le temps de récupération d'une ressource en sauvegardant une copie de celle-ci localement. Ainsi, lors de la demande d'une page :

- Soit elle n'est pas en cache : une requête est effectuée au serveur, une copie de la réponse est copiée dans le cache, et la réponse est envoyée au client.
- Soit elle est en cache : la copie précédemment créée est envoyée au client en guise de réponse. Aucune requête n'aboutit au serveur.

Souvent, le client (navigateur) utilisé par l'utilisateur réserve un espace disque de la machine pour pouvoir stocker une quantité de données en cache. Ce processus permet

1. ensemble sémantique de requêtes d'un même utilisateur

d'accélérer la navigation en ne redemandant au serveur que les choses non déjà vues. En contrepartie, du côté du serveur, tout se passe comme si l'utilisateur ne redemandait jamais la même page, ce qui du point de vue du comportement n'est pas réaliste.

Un *proxy* est une machine intermédiaire placée entre le client et le serveur, qui permet d'acheminer indirectement les requêtes de l'un vers l'autre. Le proxy, lorsqu'il est couplé au cache, corrompt encore plus les données.

Lorsqu'il existe un cache sur le proxy, on parle de cache global : en effet, les caches locaux permettent d'économiser du trafic réseau pour un utilisateur faisant plusieurs fois la même requête, les caches globaux étendent ce concept à toutes les requêtes de tous les utilisateurs des machines clientes du proxy. On considère qu'un cache est performant à partir de 15% de débit sauvegardé. Usuellement, le taux de requêtes perdues par les serveurs à causes de proxy-cache est de l'ordre de 40%. Dans la réalité, les réseaux sont conçus de manières hiérarchiques, et il y a souvent plusieurs machines intermédiaires entre clients et serveur.

En résumé, deux types de données peuvent être perdus : les requêtes manquantes car déjà en cache, les adresses de provenance erronée car correspondant à des requêtes passées via un proxy.

Sur une suite de requêtes simples nous pouvons perdre une grande partie des données initiales ; nous présentons dans la section suivante la reconstruction des données.

3 Reconstruction des données

Plusieurs travaux parlent du pré-traitement de données et de la reconstruction des données manquantes. Pour aller plus loin, nous avons voulu estimer empiriquement quelle quantité de données notre reconstruction permettait de retrouver.

Pour cela, nous nous plaçons du côté serveur et nous supposons la connaissance du site web. Nous modélisons ce dernier par un graphe où les nœuds représentent les pages et les arcs, les liens hypertextes. L'analyse des fichiers de logs se fait chronologiquement, le but étant pour chaque enregistrement, de l'associer à la bonne visite utilisateur. Pour cela, nous faisons des distinctions sur les adresses de provenance de la requête, sur le type de client utilisé (s'il est renseigné), et enfin nous détectons les incohérences dans la navigation. La détection de ces incohérences requiert la mémorisation de l'historique de chaque visite en cours. Soit s une visite ouverte, on note $h_s(i)$ la $i^{\text{iéme}}$ page dans l'historique de la visite s, i.e. $h_0(1)$ correspond à la dernière page vue par l'utilisateur associé à la visite 0. Nous présentons avec la fonction Associe comment

1. nous associons l'enregistrement courant $h(0)$ à la visite la mieux adaptée ;
2. nous corrigeons le fichier de logs.

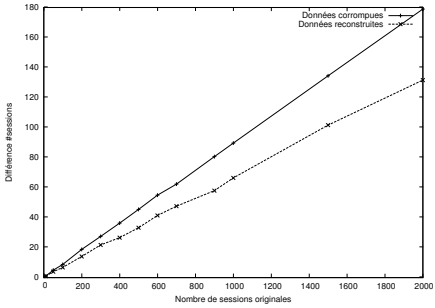
Pour résumer, nous associons au log courant la visite la plus adaptée, soit ayant un lien direct depuis la dernière page vue, soit dont la remontée dans l'historique pour trouver une page compatible est la plus courte et dans ce dernier cas, nous ajoutons la liste des logs de l'historique comme logs manquants.

Pour pouvoir évaluer la pertinence de la reconstruction des logs avec cette méthode, nous avons mené une série d'expérimentations sur des données artificielles. Des pages d'un site web, nous générons des navigations virtuelles. Nous sélectionnons une page

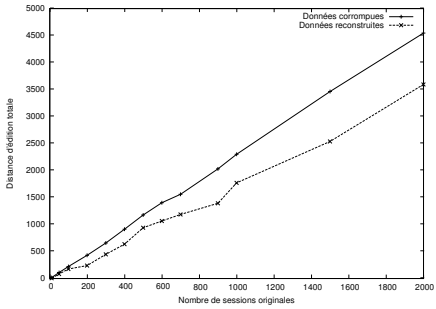
Fonction `Associe`.
Data : $h(0)$ //le log à classer
pour chaque $k \in$ *visitess_ouvertes* **faire**
  si $h_k(1) \rightarrow h(0)$ **alors**
    `Associe`(k, $h(0)$) /*associe $h(0)$ avec la visite k */
  finsi
finprch
min_historique $= \infty$;
pour chaque $k \in$ *visitess_ouvertes* **faire**
  pour $2 \leq i \leq |h_k|$ **faire**
    si $h_k(i) \rightarrow h(0)$ **alors**
      si *min_historique* $> i$ **alors**
        min_historique $= i$;
        min_visite $= k$; break;
      finsi
    finsi
  finpour
finprch
si *min_historique* $< \infty$ **alors**
  /*inconsistance trouvée : il faut ajouter ce qui manque */
  `Ajouter_Logs_Manquants`(*min_visite, min_historique*);
  `Associe`(*min_visite*,$h(0)$);
sinon
  /*pas d'historique pour se raccrocher : nouvelle visite */
  $l =$ `Nouvelle_Visite()`;
  `Associe`(l,$h(0)$);
finsi

aléatoirement dans le site, et d'une page initiale nous calculons un plus court chemin, que nous adaptons ensuite avec la possibilité de revenir en arrière ou de ne pas prendre la bonne direction. Les visites utilisateurs obtenues sont donc celles de référence. Nous simulons ensuite l'existence de proxy-cache sur les enregistrements pour obtenir un fichier de logs incomplet. Nous extrayons là encore des visites sans la détection des incohérences. Puis nous appliquons notre méthode sur les même données. Les nombres de visites détectées lors des phases « dégradée » et « dégradée et reconstruite » sont différents. Pour pouvoir comparer les deux ensembles de visites extraites nous calculons la différence de visites extraites correctement, d'une part, puis après ré-organisation des logs suivant leur visite originale, nous calculons une distance d'édition (Levenshtein, 1966) pour comparer chaque visite individuellement. La distance que nous donnons ici est la somme des distances pour toutes les visites. Les résultats sont présentés en Fig. 1.

Les expérimentations ont été menées sur 6 ensembles de logs artificiels différents. Les deux indicateurs de qualité se comportent bien : en effet, le nombre de changement de visites est plus faible dans la partie reconstruite et la distance d'édition entre les visites prises deux à deux est aussi plus faible. Ceci veut dire que la reconstruction



(a) Différence #visites.



(b) Distance d'édition totale.

FIG. 1 – *Influence de la reconstruction des données.*

permet de détecter plus de visites, et que les visites reconstruites sont plus proches des visites originales. Les données ainsi extraites sont donc plus proches des données de référence, ce qui implique des résultats meilleurs en apprentissage.

Les résultats présentés sont simulés sur 10 machines dont 2 derrière le même proxy. En faisant augmenter ce dernier nombre, le gain en distance devient plus faible mais reste strictement positif. Dans le même temps, le gain sur le nombre de visites décroit mais beaucoup moins rapidement.

L'inférence grammaticale peut remplir la tâche d'apprentissage dans le processus d'extraction de connaissances, cette technique est connue pour une faible robustesse face au bruit. Nous travaillons actuellement à mettre en place un protocole d'expérimentations permettant de montrer le réel impact du bruit du aux réseaux dans les données de type web. Des travaux (Frias-Martinez 2002, Gery 2003), utilisent des séquences fréquentes comme support à la prédiction de nouvelles pages demandées dans la navigation d'un utilisateur. Nous voulons apprendre de notre côté des modèles structurés stochastiques (automates finis déterministes) et comparer les résultats obtenus avec la méthode d'extraction de séquences fréquentes largement répandue. Nous pensons que les automates appris peuvent être plus précis en prédiction que les simples modèles de séquences, mais sont moins robustes si les données en entrée sont peu fiables.

4 Conclusion

Nous avons présenté dans cet article une méthode de reconstruction de données que nous avons évaluée sur des données artificielles. Les résultats obtenus, bien que préliminaires, sont encourageants. Nous voulons continuer dans ce domaine en comparant deux méthodes d'apprentissage sur les données brutes et celles reconstruites pour montrer que l'inférence grammaticale peut être utilisée sur ce type de données avec de bons résultats.

Références

Bidel S., Lemoine L., Piat F., Artières T. et Gallinary P. (1999), Statistical Machine Learning for Tracking Hypermedia User Behaviour, MLIRUM.

Borges J. et Levene M. (1999), Data Mining of User Navigation Patterns, WEBKDD'99.

Catledge L.D. et Pitkow J.E. (1995), Characterizing Browsing Strategies in the World-Wide Web, Computer Networks and ISDN Systems, 27(6), pp 1065–1073.

Chevalier K., Bothorel C., Corruble V. (2003), Discovering Rich Navigation Pattern on a Web Site in Discovery Science, Proc. of 6th International Conference, DS 2003.

Cooley R., Mobasher B. et Srivastava J. (1999), Data Preparation for Mining World Wide Web Browsing Patterns, Knowledge and Information Systems, 1(1), pp 5–32.

Cooley R., Srivastava J. et Mobasher B. (1997), WEB Mining: Information and Pattern Discovery on the WWW, Proceedings of ICTAI'97.

Frias-Martinez E. et Karamcheti V. (2002), A Prediction Model for User Access Sequences, WEBKDD Workshop: Web Mining for Usage Patterns and User Profiles.

Gery M. et Haddad H. (2003), Evaluation of Web Usage Mining Approaches for User's Next Request Prediction, Proceedings of WIDM'03.

Karampatziakis N., Paliouras G., Pierrakos D., Stamatopoulos P. (2004) Navigation Pattern Discovery Using Grammatical Inference in Grammatical Inference: Algorithms and Applications, Proc. of ICGI 2004.

Levenshtein V.I. (1966), Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics–Doklady, 6:707-710, 1966.

Luotonen A. (1995), The Common Log File Format, W3C Recommandation, *http://-www.w3.org/Daemon/User/Config/Logging.html*.

Pitkow J.E. (1997), In Search of Reliable Usage Data on the WWW, Proceedings of the Sixth International WWW Conference.

Pitkow J.E. et Pirolli P. (1999), Mining Longest Repeating Subsequences to Predict WWW Surfing, Proc. of USITS'99.

Tanasa D. et Trousse B. (2004), Advanced Data Preprocessing for Intersites Web Usage Mining, IEEE Intelligent Systems, Vol. 19(2):59–65.

Summary

Web Usage Mining is used to detect types of users during an internet navigation. This task can be viewed as Knowledge Discovering process: some steps are required in order to accomplish the whole process. Data used in this kind of task is almost always log server files. Data pre-processing is the first step, so it is an important one because of its outcome on the next learning step. Some works show different methods to extract web data, and recently structured models have been used to learn users navigation models. We propose in this paper to continue research on Web Usage Mining using Grammatical Inference. To avoid weak robustness, we need to use reliable data. We present here methods which process raw data and create reliable users visits. Finally, we describe an experimental protocol in order to show why pre-processing is important when the learning method is Grammatical Inference.