Tressel: Semantic mark-up of RSS feeds

Brian McLernon Nicholas Kushmerick
School of Computer Science and Informatics
University College Dublin, Ireland
{brian.mclernon,nick}@ucd.ie
www.smi.ucd.ie/tressel

ABSTRACT

The recent explosion in the popularity of RSS and other syndication technologies has led to a wealth of information being published from an increasingly diverse range of sources. However this popularity makes it difficult for users to find interesting documents, and this challenge is compounded by the fact that most RSS clients offer very modest personalization capabilities.

We propose Tressel, a collaborative RSS aggregator that extracts semantically meaningful passages of text from RSS feeds. Tressel employs a semi-supervised machine learning algorithm to identify semantic information. The algorithm learns from a small amount of training data provided by the user. Tressel is collaborative in that the input of each user is used to benefit all of the users.

One challenge with such an open system is the danger that users could (intentionally or accidentally) introduce noisy training data. In this paper, we describe Tressel's architecture and adaptive information extraction algorithm, and then report on experiments which demonstrate that we can reliably detect noisy training data.

1. INTRODUCTION

RSS [web.resource.org/rss] and similar syndication technologies have been available for years, however, their use has recently skyrocketed due primarily to wide scale adoption by blogging and news sites. The large variety of RSS feeds, clients and aggregators means that user's have ready access to ever larger document streams. However, most RSS clients offer relatively poor support for personalized retrieval or filtering, forcing users to manually sift through vast quantities of text to find information pertinent to their need.

In response to this challenge, we propose the Tressel RSS aggregator. Tressel uses adaptive information extraction [3] techniques to identify document fragments that summarize the document's semantic content. This extracted data can then be used to drive sophisticated semantic filtering capabilities, as well as to highlight parts of the text that are of interest to the user.

There has been considerable commercial interest in collaborative document tagging (eg, del.ic.ious, flickr, etc). Tressel takes this idea one step further: rather than allowing users to annotate documents as a whole, users can share annotations of particular document fragments.

Tressel is intended as a wide-scale collaborative RSS aggregator. As described below, the data extraction component involves learning extraction patterns from manually annotated training data. Naturally, a large group of users will be interested in extracting a wide variety of data from many distinct RSS feeds. Our conjecture is that in a sufficiently broad user community, there will be some overlap in users' interests. Consequently, a key advantage of Tressel compared to conventional adaptive information extraction scenarios is that the effort of providing annotated training data is amortized over a large number of users, with each individual user only needing to provide a few annotations (say, one document's worth).

Fig. 1 shows an example scenario of Tressel in action. Fig. 1(a) depicts a user A's annotations of a document. When a second user B views another document (Fig. 1(b)), he benefits from the system learning from A's annotations. If he spots an error (in this case, B notices that Buffett's first name was not annotated), B can edit the annotation. The system learns from the update and applies this to subsequent annotations (Fig. 1(c)). The power of Tressel is that these refined annotations are supplied to a pattern learning algorithm so that new documents (perhaps from different RSS feeds) can also be annotated.

2. ARCHITECTURE

As shown in Fig. 2, Tressel comprises three main components. The Poller is the core RSS engine that monitors the feeds and maintains a local cache of their content. Tplex is the information extraction component which learns extraction patterns from training data and then applies these rules to the cached RSS documents. Finally, a user interface allows users to view the RSS feeds (filtered by, and augmented with, the semantic text fragments identified by Tplex), and to add semantic annotations of there own to documents.

Once an RSS feed has been registered with the Poller, it is regularly polled for updates and any new documents are retrieved and cached. Thus Tressel always maintains an up-to-date local cache of the feed. With this cache, Tressel allows users to view the feed's documents and (optionally) provide document annotations. Tressel is flexible and open in that users can create new fields on the fly, which are then automatically shared with other users. For example, a user might decide to annotate the names of CEOs in a few documents from a financial news feed. All users of the system that subscribe to that feed can then see and edit these annotations.

Once Tressel has an initial set of document annotations, it gives them to the Tplex algorithm (described below) which

WWW2006, May 22–26, 2006, Edinburgh, UK.
.

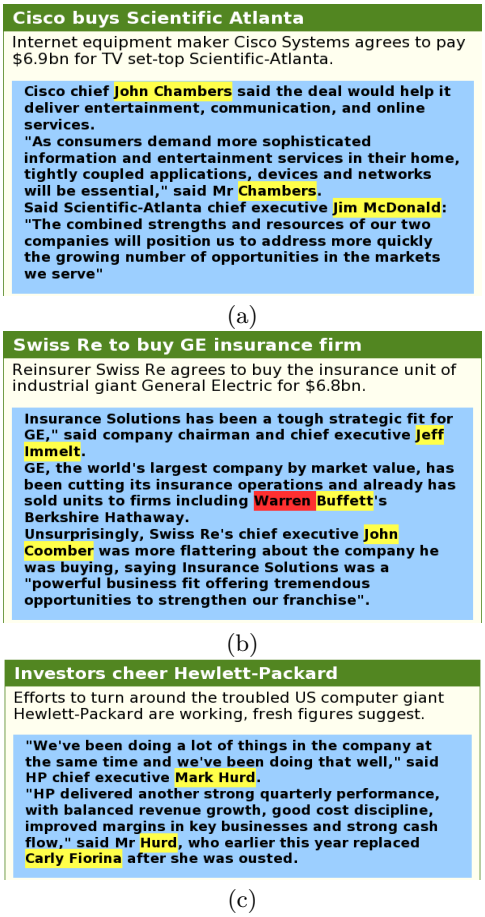

Figure 1: Users annotate documents (a). Tressel learns from annotated documents and users can edit annotation errors (b). Tressel learns from collaboratively annotations and applies this to subsequent documents (c).

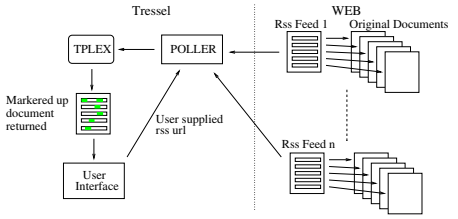

Figure 2: An overview of the Tressel system.

learns extraction rules that can be applied to the rest of the documents in the feed. The learned rules can then be applied to new feed documents, and those fragments extracted with high confidence are highlighted for subsequent viewers. If a user thinks that a fragment has been incorrectly highlighted they can edit the annotation and submit the changes. The learning algorithm then takes these changes into account on the next pass over the feed.

3. THE TPLEX ALGORITHM

Tplex [4] is a transductive algorithm for learning information extraction patterns. A key distinction compared to other adaptive information extraction algorithms (eg, [1, 2]) is that Tplex learns from a mixture of manually labelled as well as unlabelled documents. The key idea is to exploit the following recursive definitions: good patterns extract good fragments, and good fragments are extracted by good patterns.

Tplex operates by bootstrapping the learning process from a seed set of labeled examples. The examples are used to populate initial pattern sets for each target field, with patterns that match the start and end positions of the seed fragments. Each pattern is then repeatedly generalized and matched against the corpus to produce more patterns.

The pattern and fragment scores are initialized from the labeled data. They are then updated based on a recursive definition of "goodness". This process iterates until the scores have converged. Our scoring mechanism calculates the "goodness" score of a pattern as a function of the scores of the positions that it matches, and the score of a position as a function of the scores of the patterns that extract it.

Tplex is a multi-field extraction algorithm in that it extracts multiple fields simultaneously. By doing this, information learned for one field can be used to constrain patterns learned for others. Specifically, our scoring mechanism ensures that if a pattern scores highly for one field, its score for all other fields is reduced. In so doing this bias is then transferred to the positions that the pattern extracts. Note that Tressel continually resamples the feeds and re-learns the patterns. In this way the system can adapt to changes in the formatting, style or content of a feed over time.

We will now describe in detail the operation of the Tplex algorithm. Positions are denoted by r, and patterns are denoted by p. Formally, a pattern is equivalent to the set of positions that it extracts.The notation $p \rightarrow r$ indicates that pattern p matches position r. Fields are denoted by f, and F is the set of all fields.

The labelled training data consists of a set of positions $R = \{\ldots, r, \ldots\}$, and a labelling function $T : R \rightarrow F \cup \{X\}$ for each such position. $T(r) = f$ indicates that position r is labelled with field f in the training data. $T(r) = X$ means that r is not labelled in the training data (i.e. r is a negative example for all fields).

The unlabelled test data consists of an additional set of positions U. Given this notation, the learning task can be stated concisely as follows: extend the domain of T to U, i.e. generalize from $T(r)$ for $r \in R$, to $T(r)$ for $r \in U$.

3.0.1 *Initialization*

As the scores of the patterns and positions of a field are recursively dependant, we must assign initial scores to one or the other. Initially the only elements that we can classify with certainty are the seed fragments. We initialise the scoring function by assigning scores to the positions for each of the fields. In this way it is then possible to score the patterns based on these initial scores.

From the labelled training data, we derive the prior probability $\pi(f)$ that a randomly selected position belongs to field $f \in F$:

$$\pi(f) = |\{r \in R \,|\, T(r) = f\}|/|R|.$$

Given the priors $\pi(f)$, we score each potential position r

in field f:

$$\text{score}_f^0(r) = \begin{cases} \pi(f) & \text{if } r \in U, \\ 1 & \text{if } r \in R \wedge T(r) = f, \text{ and} \\ 0 & \text{if } r \in R \wedge T(r) \neq f. \end{cases}$$

The first case handles positions in the unlabelled documents; at this point we don't know anything about them and so fall back to the prior probabilities. The second and third cases handle positions in the seed documents, for which we have complete information.

3.0.2 Iteration

After initializing the scores of the positions, we begin the iterative process of scoring the patterns and the positions. To compute the score of a pattern p for field f we compute a positive score, $\text{pos}_f(p)$; a negative score, $\text{neg}_f(p)$; and an unknown score, $\text{unk}(p)$. These quantities are defined as follows for each field f and pattern p:

$$\text{pos}_f(p) = \frac{1}{Z_p} \sum_{p \to r} \text{score}_f^t(r),$$

where $Z_p = \sum_f \sum_{p \to r} \text{score}_f^t(r)$ is a normalizing constant to ensure that $\sum_f \text{pos}_f(p) = 1$.

$$\text{neg}_f(p) = 1 - \text{pos}_f(p).$$

$$\text{unk}(p) = \frac{1}{|\{p \to r\}|} \sum_{p \to r} \text{unk}(r),$$

where $\text{unk}(r)$ measures the degree to which position r is unknown. To be completely ignorant of a position's field is to fall back on the prior field probabilities $\pi(f)$. Therefore, we calculate $\text{unk}(r)$ by computing the sum of squared differences between $\text{score}_f^t(r)$ and $\pi(f)$:

$$\begin{aligned} \text{unk}(r) &= 1 - \frac{1}{Z}\text{SSD}(r), \\ \text{SSD}(r) &= \sum_f \left(\text{score}_f^t(r) - \pi(f)\right)^2, \\ Z &= \max_r \text{SSD}(r). \end{aligned}$$

The normalization constant Z ensures that $\text{unk}(r) = 0$ for the position r whose scores are the most different from the priors—ie, r is the "least unknown" position.

For each field f and pattern p, $\text{score}_f^t(p)$ is defined in terms of $\text{pos}_f(p)$, $\text{neg}_f(p)$ and $\text{unk}(p)$ as follows:

$$\text{score}_f^{t+1}(p) = \frac{\text{pos}_f(p)}{\text{pos}_f(p)+\text{neg}_f(p)+\text{unk}(p)} \cdot \text{pos}_f(p)$$

This definition penalizes patterns that are either inaccurate or have low coverage. Finally, we complete the iterative step by calculating a revised score for each position:

$$\text{score}_f^{t+1}(r) = \begin{cases} \text{score}_f^t(r) & \text{if } r \in R \\ \frac{\sum_{p \to r} \text{score}_f^t(p) - \min}{\max - \min} & \text{if } r \in U, \end{cases}$$

where $\min = \min_{f,p \to r} \sum_{p \to r} \text{score}_f^t(p)$ and $\max = \max_{f,p \to r} \sum_{p \to r} \text{score}_f^t(p)$, are used to normalize the scores to ensure that the scores of unlabelled positions never exceed the scores of labelled positions. The first case in the function for $\text{score}_f^{t+1}(r)$ handles positive and negative seeds (i.e. positions in labelled texts), the second case is for unlabelled positions. We iterate this procedure until the scores of the patterns and positions converge.

4. AUTOMATIC MODERATOR

Tressel's open approach to document annotation gives rise to potential problems. There is always a possibility that a user will (intentionally or accidentally) provide incorrect document annotations. As a semi-supervised learning algorithm, Tplex is quite sensitive to noisy training data. Thus, it is necessary for Tressel to validate the annotations supplied by the users that are used for input to the algorithm.

One approach to this would be to compare the annotations of multiple users on the same document and come to a consensus. However, collaborative systems have always be subject to sparcity of information and there is no guarantee that a document will be annotated by more than one user. And even if multiple annotated examples exist, it might be difficult to reach a correct consensus about the annotations if there are inconsistencies in a relatively small set.

Therefore, Tressel includes as part of its core functionality a method for automatically detecting noisy annotations that avoids the need to have multiple submissions of a particular document. It does this by comparing a documents' annotations with those of other documents via Tplex.

Tressel assumes that the user community consists of a (small) trusted group and a (large) untrusted group. Annotations from untrusted users are checked and rejected if found to be noisy. The key idea of our noise detection algorithm is to train Tplex with and without each suspect document in turn, and compare the results.

To do this we select a set of annotated documents from trusted users and a document from an untrusted user. We then train Tplex on one document at a time and tag the other documents in the set. An accuracy score is determined based on the similarity between the fragments suggested by Tplex and the original tagged fragments. Thus we generate a score for each document against the set. Typically the trusted documents will score highly. If the score from the untrusted document is within the range of the scores from the trusted documents then the annotations are accepted. However, if its score falls below this range, then the document is considered a rogue. Once a rogue document has been detected, Tressel automatically corrects the annotations by tagging the fragments suggested by Tplex when trained on the trusted documents.

In this way we can track the accuracy of a particular user. Over time the trust level of a user will increase if they consistently provide accurate annotations. Thus they can migrate from the untrusted set to the trusted set. But should the quality of their submissions fall they will be relegated back to the untrusted set.

5. EXPERIMENTS

We carried out a number of experiments to test the accuracy with which we could detect and correct errors in the inputs given to the system. To do this we simulated a case where a user incorrectly annotates a document, by subjecting it to one of three types of alteration. The first type of alteration involves randomly moving tag pairs by up to

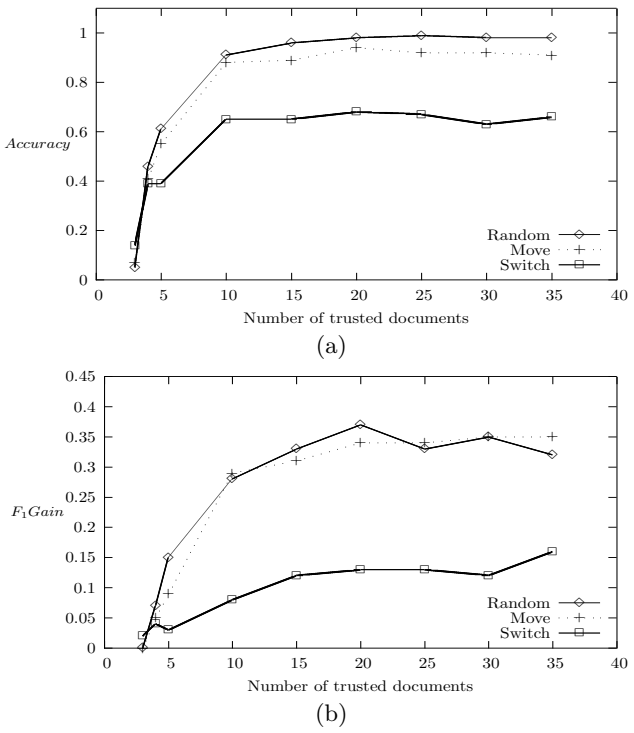

Figure 3: Rogue document detection accuracy (a) and F1 gain after retagging documents (b).

four places. The second involves swapping tag pairs with other tags and the third altered the annotations by randomly re-tagging the document. We varied the number of trusted documents provided to the system from 3 to 35. We then measured the fraction of time we can detect the rogue document, and the gain in F1 accuracy of the re-annotated document compared to the original document. The averaged results of 20 iterations are presented in Fig. 3.

The results in Fig. 3(a) show that even with quite few trusted documents, Tressel can reliably identify a rogue document, though the accuracy depends on how the document was altered. Specifically, when the document was tagged randomly the system correctly identified the rogue document 98% of the time given just 20 trusted documents. Similarly when the tags were moved the system was able to identify the correct document 94% of the time with 20 trusted documents. The switch alteration (a correctly annotated document with mislabelled fragments) is the hardest to detect. Here the system identified the rogue document 68% of the time when given 20 trusted documents. The poorer performance on this task is due to a high degree of similarity between two of the four test fields.

Fig. 3(b) shows that by re-annotating the identified document we can achieve quite substantial increases in extraction F1. As before the performance is dependant on the mode of alteration. As the system automatically re-annotates those documents that it perceives to be rogue, if it identifies the wrong document it will incur a negative F1 gain. Even so the system generated gains in F1 for all types of alteration. When random tagging was used the system recorded a 37% gain in F1 with 20 trusted documents. And when the tags were moved there was a 34% gain in F1 with 20 trusted documents. Finally when the tags of the rogue document were switched the system achieved an F1 gain of 13% given 20 trusted documents.

6. DISCUSSION

Tressel is a new kind of semantic RSS aggregator that makes it easy for users to share document annotations, and then uses adaptive information extraction methods to learn patterns for annotating new documents.

A prototype of Tressel is currently under construction. The Poller and Tplex components are completed, and we are currently developing the user interface. Once completed, we intend to engage in large-scale user studies to evaluate its effectiveness. We believe that Tressel promises to radically change the way people use RSS, but to do so will require achieving a critical mass of users. We are therefore focusing on a simple interface and highly reliable service in order to attract as many users as possible.

7. FUTURE WORK

In addition to completing our prototype, we intend to extend our ideas in various ways. First, we believe that the underlying Tplex extraction algorithm can be made more accurate. Second, so far we have focused on extracting document fragments, but it may also be useful to provide text classification or collaborative filtering capabilities for documents as a whole.

In addition as Tressel grows it would be relatively straightforward to implement a recommender system for feeds and introduce users to new feeds that they might appreciate.

For a system such as Tressel to work successfully user interaction is essential. Therefore, to promote participation, we suggest giving the users a financial incentive. If we were to host advertisements on the Tressel main page then the users could be paid for participation. When a user supplies accurate annotations to the system they will receive a small payment derived from the advertising revenue. This will not only encourage the users to interact with the system but it will also encourage them to do so reliably.

Acknowledgements. This research was supported by grants SFI/01/F.1/C015 from Science Foundation Ireland, and N00014-03-1-0274 from the US Office of Naval Research.

8. REFERENCES

[1] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proc. Int. J. Conf. Artificial Intelligence*, 2001.

[2] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proc. American Nat. Conf. Artificial Intelligence*, 2000.

[3] N. Kushmerick and B. Thomas. Adaptive information extraction: Core technologies for information agents. *Lecture Notes in Computer Science*, 2586, 2003. Intelligent information agents: The AgentLink perspective; M. Klusch, S. Bergamaschi, P. Edwards and P. Petta, editors.

[4] B. McLernon and N. Kushmerick. Transducive pattern learning for information extraction. Adaptive Text Extraction and Mining workshop, 11th conference of the European Chapter of the Association of Computational Linguistics, 2006.