A study of machine learning methods for detecting user interest during web sessions

Alípio M. Jorge
LIAAD - INESC TEC, DCC - FCUP, Universidade do Porto, Portugal
amjorge@fc.up.pt

José Paulo Leal
CRACS - INESC TEC, DCC - FCUP, Universidade do Porto, Portugal
zp@dcc.fc.up.pt

Sarabjot S. Anand
Algorithmic Insight, India
ssanand@a-insight.com

Hugo Dias
DCC - FCUP, Universidade do Porto, Portugal

ABSTRACT

The ability to have an automated real time detection of user interest during a web session is very appealing and can be very useful for a number of web intelligence applications. Low level interaction events associated with user interest manifestations form the basis of user interest models. However such data sets present a number of challenges from a machine learning perspective, including the level of noise in the data and class imbalance (given that the majority of content will not be of interest to a user). In this paper we evaluate a large number of machine learning techniques aimed at learning from class imbalanced data using two data sets collected from a real user study. We use the AUC, recall, precision and model complexity to compare the relative merits of these techniques and conclude that useful models with AUC above 0.8 can be obtained using a mix of sampling and cost based methods. Ensemble models can provide further accuracy but make deployment more complex.

Keywords

modeling, user interaction, user interest, machine learning, imbalanced classification

1. INTRODUCTION

The automatic detection of user interest while browsing is a very appealing ability of a web client. The browser may not only detect when the user is interested but also in what region of the page (where) the user is interested in. This opens up to many possible applications such as triggering recommendations when the user seems to be interested in a particular portion of text or image. The observation of the user behavior is based on low level client side interaction events with the mouse and keyboard. The browser keeps collecting these events as the user interacts. Such a stream is fed to a classification model, embedded in the code of the web page, which decides whether the user is showing interest or not. The result of the model can then be used on the client side or transmitted to a remote server for a more demanding request.

IDEAS14 July 07-09 2014, Porto, Portugal

http://dx.doi.org/10.1145/2628194.2628197.

In our approach we use machine learning for obtaining the classification models automatically. Detecting user interest is modeled as a binary classification problem: interested/not interested. As expected, the proportion of classes is very imbalanced, given that interest events are relatively rare. Most of the time the user is simply browsing or is in an idle state. Therefore, one important challenge is how to robustly deal with the class imbalance. Other challenges, not addressed in this paper, are: how to collect the training data and how to pre-process the data in order to favor prediction.

In this paper we study the applicability of a large number of machine learning techniques for addressing class imbalance bearing in mind two aims. One is to have models that can detect a good number of interest situations with a proportion of false alarms as low as possible and the other is to obtain models that are lightly embeddable in the page and can run on the client side. The methods are evaluated on two data sets collected by us from real interaction data. We use 10-fold cross-validation and estimate the performance of the models using metrics such as the Area Under the Curve (AUC) [1], recall, precision and model size. From the empirical results we can rank the methods to be used for the purpose of detecting user's interest.

2. MODELING USER INTERACTION

As soon as web browsers and hypertext became popular, researchers started trying to model interaction behavior. Letizia, for example, was a browser agent proposed by Lieberman [2]. It observed the browsing behavior of a particular user and learned to anticipate which pages would be of interest. It recorded actions on the document such as saving or hyperlink clicking and scored words in it. These were used to recommend pages containing highly scored words to be followed next. Letizia was implemented as what would nowadays be an applet.

Others have also exploited the recordable interaction provided by the new media. Claypool et al. [3] would see some

human identified patterns of activity as *implicit interest indicators*. The user would provide, without noticing, information about his/her interest on pages solely by browsing. Not explicitly asking the user's opinion is also called *unobtrusive observation* by Goecks and Shavlik [4] who built a browser agent that collected interaction data and trained a neural network to predict the amount of interaction activity given the words of a page. This model can be used to automatically look for pages of interest to the user. These approaches for modeling user behavior do not rely on the traditional server-side clickstream data, but rather follow the philosophy of the *curious browser* [3] by collecting *client-side* data [5], also called *subsymbolic user-behavior* [6].

Browsing models may also attempt to determine, at any given instant a *browsing state*. Syeda-Mahmood and Ponceleon [7] have proposed 10 different states for users browsing videos, such as: curious, aimless browse, found something interesting, undetermined, etc. These authors have also exploited the interdependence of subsequent states using Hidden Markov Models.

Most of the above mentioned works studied or modeled user's interest on whole web pages or documents. Hijikata [8] tried to improve the user interest model on pages/words by focusing on parts of the browsed pages. These relevant parts were identified through specific interaction events such as text selection, link clicking and pointing, moving the mouse over the text while reading, etc. Although some of these actions are not necessary for browsing, they were unconsciously performed by users and carried interesting information. Hauger and Velsen [5] collected client-side interaction events for trying to identify which parts of a page had been read. They organized a controlled experiment with 53 users who had to fill in a questionnaire saying which parts of the page they actually read. This data was then used to train machine learning classification methods. Yang et al. [9] used a similar approach on small screen devices for identifying what they call *interest blocks*. Later, Hauger et al. proposed the use of eye-tracking, in addition to interaction data, and concluded that interactive behavior can be used to approximately determine gaze position [10].

2.1 Our approach

This paper is part of a larger endeavour for designing an end-to-end process for modeling user interest and deploying such models so that they can be used in real time. The process has two phases: *learning* and *deployment*. In the first phase we collect data and obtain a model. In the second we embed the model in the page and use it.

The learning phase has the following steps:

1. Select a web page as target;
2. Define an area of interest;
3. Define a question whose answer is in the defined area of interest;
4. Invite users to open the page and answer the question (as in a quiz) - users are not given any other information;
5. Collect user client-side interaction data (mouse and keyboard) during the quiz sessions;
6. Associate each interaction event (e.g. click or mousemove) to an x, y location in the page, if possible;
7. Label each event location as in the area of interest ("INT") or outside of it ("BROWSE"):
8. Aggregate events in time contiguous segments;
9. Aggregate labels in the same segment to obtain a segment label;
10. Setup a classification data set from segments;
11. Build a classification model from the data set.

After learning we deploy the model embedded in the page.

1. Embed the model in the page as JavaScript;
2. Aggregate user interaction events on the client side as a sliding active segment;
3. Invoke the model upon every event using the active segment and obtain an interest label;
4. Use the interest label for recommendation or for collecting high level interaction events.

We currently have a complete solution for the whole process but many parts of the process present interesting challenges.

2.2 Data Collection

The data to collect will be made of pairs $< Bhv, St >$, where Bhv represents the behavior of the user, captured as an account of interaction events and St is the browsing state. An important problem is how to obtain such data. So far, this has been solved by setting up a controlled user experiment or by assuming a simplified user interest model. In the first case, after browsing, users are directly asked to declare their explicit interest in a page or part of page [5, 3, 8, 7] or users are monitored using a eye-tracking device [10]. Simplified manual user interest models have been used for labeling in [2] and in [9].

The "tell me what you're thinking" approach greatly reduces the potential for acquiring data from a large number of users. In this setting, the labeling effort is high and user must complete the tedious task of answering "Did you like it?" questionnaires. One of our contributions in this paper is an experiment that enables the indirect gathering of labelled interaction data.

We have setup web based quizzes that were publicized in social networks and through email. As a result we had a very relevant number of users providing data without knowing exactly what it was for, which minimized biased behavior. In particular, we mirrored 2 relatively long web pages: one from Wikipedia on the Olympic Committee and the other from Facebook on Soccer. For each of them we proposed a question. Upon entering the page the user would read the question in a pop-window, minimize the pop-up, look for the answer in the page (text find was not particularly useful since we avoided searchable words in the question), reopening the pop-up, and choose the right option as an answer. Fig 1 illustrates one of the quizzes.

The interaction data collected included the usual graphical interface events as detailed in Table 1. All the low level events are recorded during the experiment. It is also recorded the answer given to the quiz. This generates a lot of traffic that is totally avoided at production time, as will be later detailed.

Figure 1: Screenshot of one of the pop-up quizzes as it appears to the user.

2.3 Data pre-processing

Raw interaction data is divided as a set of sessions. Each session is a stream of events of a certain type (e.g. mouse click, scroll) and at a certain time. Some events also occur at coordinates $< x, y >$. These coordinates are absolute with respect to the whole page but may differ from session to session. Some events also generate specific information (e.g. which key was stroke) which is also stored. In Fig. 2 we can see that the most frequent interaction events are, by far, mousemove, scroll, mouseover and mouseout. These statistics were calculated from the whole set of sessions.

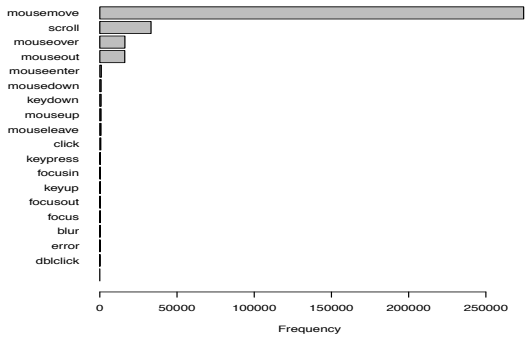


Figure 2: Frequency of events per type.

Since we know where the region of interest is (this information is automatically stored for each particular session) we can label each event as being in the region of interest (INT) or not (BROW). This can only be done, however, for the events with a defined location. In that case we compare the $< x, y >$ coordinates of the event with the defining points of the bounding box of the region of interest $< x_{top-left}, y_{top-left} >$ and $< x_{bottom-right}, y_{bttom-right} >$. Events without a spatial location are also labeled as "BROW", since there is no immediate evidence that they are in the region of interest. We have divided the sessions according to the two quizzes which will originate two data sets.

Next, for each session, we divide the time line in chunks of fixed length. For each chunk we aggregate the events of each type by counting the occurrences. For example, for the type of event "mouseclick" we have have an attribute for the chunk whose value is the number of mouseclicks in the chunk. We also measure the *xrange* and the *yrange* as the maximal difference in the normalized (with respect to the page size) x and y coordinates, respectively, in the time chunk. Therefore, we have 20 variables (attributes): one for each event type and the two ranges. Finally, we label each chunk as "INT" if at least one of the events in the chunk is labeled as such. All other chunks are labeled as "BROW". In our experiment the duration of the chunk was set to 5 seconds. Alternative values can be considered, though.

The x and y spatial coordinates that are associated with some of the events are session dependent and must be normalized. So, instead of using the recorded coordinates we use relative values calculated as $x/pagewidth$ and $y/pagelength$. The two resulting data sets "Olympic" and "Soccer" are summarized in Table 2.

Table 1: Type of interaction events collected. "Y" means that the x, y location of the event is recorded. Some event types have the same description plus a (*) mark but there may be differences which are not detailed in this paper.

Event type	x, y	Description
mouseover	Y	mouse enters an element.
mouseenter	Y	mouse enters an element(*).
mousemove	Y	mouse is moved.
mouseout	Y	mouse leaves an element.
mouseleave	Y	mouse leaves an element(*).
blur		element loses the focus.
focusout		element loses the focus(*).
focus		element receives the focus.
focusin		element receives the focus(*).
scroll		something is scrolled.
mousedown	Y	mouse button pressed.
mouseup	Y	mouse button released.
click	Y	mousedown and mouseup events occur on the same element OR an element is activated by the keyboard.
keydown	Y	key is pressed.
keypress	Y	keystroke leads to a character being added to an HTML element.
keyup	Y	key is released.
error		browser encounters a JavaScript error or a non-existing image file.
dblclick	Y	two click events take place on the same element within a reasonable timeframe.

Table 2: The two resulting data sets.

Data set	Size	#BROW	#INT	%BROW	%INT
Olympic	1994	1789	205	90%	10%
Soccer	1610	1564	46	97%	3%

3. IMBALANCED CLASSIFICATION

In this paper we specifically focus on step 11 of the learning phase (building a classification model). Our purpose here is to study robust machine learning methods that can cope with the generated data. One very distinct feature of this data is its class imbalance. The number of segments

where the user is going to show interest is relatively low (10% and 3% in our experiments). This requires the use of algorithms for imbalanced data as well as an adequate evaluation methodology. In this section we describe the techniques used in our study.

Class Imbalance is typically addressed by data sampling. The simplest approach is to randomly undersample the majority class or randomly oversample (with replacement) the minority class. The amount of under(over)sampling depends on the domain being modeled and often does not simply translate to the generation of a uniformly distributed class label. Hence, Chawla et al. [11] proposed the use of cross-validation to determine the optimal amount of sampling.

As opposed to simple under(over)sampling, more "intelligent" sampling approaches have also been proposed. The most significant such oversampling method is Synthetic Minority Over-sampling TEchnique (SMOTE) [12]. The removal of redundant negative instances and tomek links was proposed as an alternative to undersampling [13]. In SMOTE, synthetic examples are generated from each positive training instance along the (hyper-)plane joining the positive instance to one of its k nearest positive instance neighbours. In SMOTE, synthetic examples are generated from each positive training instance, t_{i+}, (the seed instance) as follows. First the k nearest neighbours, n_{it}'s of t_{i+} are retrieved. Next r of these nearest neighbours are chosen through sampling by replacement, where r is the number of synthetic examples that each of the positive training instances will contribute to the new oversampled training data set. For example, if 300% oversampling is to be carried out then r = 3. The synthetic data instance s_{ij} is then generated as $\vec{s}_{ij} = \vec{t}_{i+} + q.(\vec{n}_{ij} - \vec{t}_{i+})$ where q is a random number between 0 and 1. Note that Oversampling with replacement is a special case of SMOTE where q is set to 0. Kubat and Matwin [13] proposed a method for undersampling the negative examples by removing redundant instances and tomek links as opposed to random undersampling. The new training set is seeded with all positive instances and a randomly selected negative instance. Using the seed instances as the model, the remaining negative instances are classified using the 1-Nearest Neighbour algorithm and misclassified negative instances are added to the new training set. Finally those negative instances with a positive instance as the nearest neighbour (referred to as tomek links) are removed.

Both, undersampling and oversampling have drawbacks. First, undersampling leads to loss of information as negative instances not used in learning could impact model performance on unseen examples. On the other hand, oversampling can result in over-fitting and an increase in learning time as the training data can increase substantially in size as a result.

More recently, repetitive sampling-based ensemble models have also been proposed. Seiffert et al. applied AdaBoost [14] to class imbalance problems and showed it to be effective in increasing the AUC [15]. Chawla et al. [16] applied SMOTE before each Adaboost iteration while Seiffert et al. [17] used undersampling, both showing positive results. Liu et al. [18] proposed EasyEnsemble that builds multiple models using different subsamples of the majority class and then computes the sum of the posterior probability of the class label assigned by the base classifiers. use of clustering to partition the majority class and then build a model for each partition along with all the instances of the minority class. Experimental validation of clustering approaches to the imbalance problem, by Molinara [19], however, suggest that random partitioning is actually superior to clustering based methods. While ensemble models address the shortcomings of undersampling by resampling the data, and hence minimizing information loss, the models generated are larger and more complex as they consist of a set of models and, potentially, a weighting associated with the base models.

4. ALGORITHMS

We evaluate six methods for dealing with class imbalance in our data. Four of these methods are sampling-based, one is cost-based and one uses a rather novel genetic programming based approach.

4.1 Sampling-based Methods

Random Over Sampling (ROS): This method generates a sample of the minority class by sampling with replacement, effectively making multiple copies of the existing instances. An r% oversample results in a data set consisting of all the majority class instances and $(1 + r/100) \times n$ minority class instances, where n is the number of minority class instances within the original data set. In our experiments we varied the oversampling parameter r from 100% to 1500% in step sizes of 200%.

Random Under Sampling (RUS): RUS reduces the number of majority class training instances by randomly removing instances belonging to the class from the training instances. The final number of instances used for training when using r% under sampling is $r \times n/100$, where n is the number of majority class instances within the original data set. In our experiments we varied the value of r from 5% to 100% in steps of 5%.

SMOTE: This is an implementation of Synthetic Minority Over-sampling TEchnique as proposed by Chawla et al [12]. It takes two parameters, the level of oversampling r and the number of neighbours, k, used for generating the synthetic instances. We experimented with values of r ranging from 100% to 1500% with step size of 200% and k taking integer values in the interval $[1, 15]$.

Tomek: This is an implementation of Kubat and Matwin's [13] approach to one-sided selection.

4.2 Genetic Programming

The Genetic Programming (GP) approach to evolutionary computing breeds a population of candidate programs to solve a problem [20]. These programs are typically represented as tree structures that are evolved over multiple iterations (generations). The initial population is generated at random while future generations are evolved through the measurement of fitness of the candidates programs and subsequently selecting candidates for reproduction with a probability proportional to their fitness. The candidate parent programs generate offsprings through the application of the *crossover* and *mutation* genetic operators.

We use a GP to generate a projection of the original data set to a feature space that is more amenable to learning a classification model in the presence of a skewed class distribution. Hence given a training data set, $D \subset \Re^n$, defined using a set of n attributes A, labelled using one of two class labels $\{L_1, L_2\}$, we aim to learn a set of k features, where each feature is a function of a subset of attributes in A and constants within some continuous range. Hence an

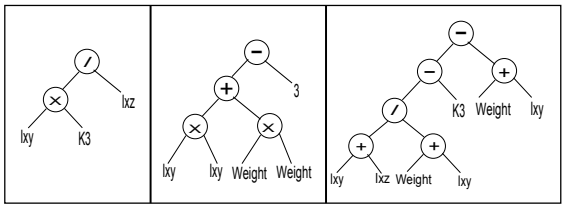


Figure 3: Example Chromosome

individual (chromosome), in our case, consists of a set of k programs, each representing a new dimension in the feature space. The function set, F, used to construct these features are the ordinary arithmetic functions {+,-,*,/} where, as is common in GPs, '/' is the protected division function that returns a value 1 when the denominator is 0. The terminals (nodes within a program tree with no children) are either attributes or constants. An example chromosome is shown in Figure 3, representing a three-dimensional feature space defined by the features $\frac{I_{XY}K_3}{I_{XZ}}$, $I_{XY}^2 + Weight^2 - 3$ and $\frac{I_{XY}+I_{XZ}}{Weight+I_{XY}} - K_3 - (Weight + I_{XY})$.

To evaluate the fitness of a chromosome, a learning algorithm is applied to the data projected to the feature space using projections as defined by the chromosome. 10-fold cross validation is used to compute a composite confusion matrix. The confusion matrix is then used to compute the chromosomes' fitness defined as the area under the ROC curve (AUC).

The population is initialized using the *Ramped half-and-half* method [21]. Chromosomes with an above average fitness within the current population are chosen to produce the next generation. This is acchieved through crossover and mutation. Subtree and gene crossover and subtree mutation are used to generate the new generation.

4.3 Cost-Based Methods

Cost-based methods assume the availability of a cost matrix defining the cost of misclassification of an instance belonging to class C_i into class C_j. Class imbalance can be viewed as a misclassification cost minimization problem by assigning a higher cost to misclassification of a positive instance as a negative instance. MetaCost [22] is a popular method for dealing with cost minimization problems. Its attractiveness stems from the fact that it can be used with any base classifier. It works by estimating the posterior probability of the class labels given a training instance and then relabels the instance so as to minimize the expected misclassification cost. The final model is learnt from the relabelled training instance.

We experimented with a number of cost matricies. For the Olympic data set we evaluated a cost of misclassifying a positive instance as a negative one of 2 through to 10 in steps of 2. For the Soccer data that have a higher skew we experimented with values ranging from 5 to 55 in steps of 5.

With each of the methods above, we used J48 (the implementation of C4.5 [23] in WEKA), JRip (the WEKA implementation of Ripper [24]), PART [25] and AdaBoost [14] to build models.

5. RESULTS

To find the optimal parameters for the individual sampling methods and estimate the expected quality of the model learnt, 10-fold cross validation was used to generate the confusion matrices for each of the four algorithms.

The cross validation was run ten times using different random seeds to account for the stochastic nature of the genetic algorithm and random sampling. The average AUC was used for selecting the best parameters for each model/sampling method.

Tables 3 and 4 summarize the results obtained. For each of the seven methods used to build models (four using sampling, 1 using the GP, 1 using MetaCost and 1 using a combination of sampling and ensemble learning), the precision and recall, using a threshold of 0.5 on the posterior probability of the class labels given an instance, estimated by the models learnt, area under the ROC curve (AUC) and the model complexity are shown. For each base algorithm (J48, Ripper and PART), we also compare the performance of the models with a base model (No Sampling) that does not apply any approach to address class imbalance.

The AUC gives us a summary of the model performance for all possible thresholds used on the posterior probability of the class labels to classify an instance as positive or negative. From the tables it is apparent that all methods for addressing the class imbalance with the exception of Random Oversampling (ROS) improve the AUC considerably over the base value (no sampling). On the whole, AdaBoost outperforms all other classifiers. The only exception being MetaCost when applied to the Soccer data with the base algorithm PART. Note however that there is no consensus as to which class imbalance method produces the best model when applied to AdaBoost,although SMOTE seems to perform better than the other approaches in general. With regard to the oversampling methods, SMOTE appears to outperform random oversampling. The latter appears to overfit both data sets and often performs worse than "No Sampling". When considering only classifiers consisting of a single model, the GP based approach using J48 performs best on the Olympic data set while MetaCost outperforms all classifiers on the Soccer data, including AdaBoost. Of the sampling based methods SMOTE and undersampling build models comparable in their AUC values and produce more accurate models than the GP or MetaCost when using Ripper on both data sets.

```
xrange <= 0.244042: BROWSE
xrange > 0.244042
|   yrange <= 0.593379
|   |   mouseout <= 6: BROWSE
|   |   mouseout > 6
|   |   |   scroll <= 0
|   |   |   |   mouseout <= 10: BROWSE
|   |   |   |   mouseout > 10: INT
|   |   |   scroll > 0: BROWSE
|   yrange > 0.593379
|   |   keyup <= 0
|   |   |   mousemove <= 12
|   |   |   |   mouseleave <= 0: INT
|   |   |   |   mouseleave > 0: BROWSE
|   |   |   mousemove > 12: INT
|   |   keyup > 0
|   |   |   mousemove <= 44: BROWSE
|   |   |   mousemove > 44
|   |   |   |   click <= 0: INT
|   |   |   |   click > 0: BROWSE
```

Figure 4: Decision tree for the Olympic data set.

Model complexity represents the size of the obtained models. It is measured as the number of conditions in the model. In the case of trees, it is the number of nodes (Fig. 4). In rules it is the number of conditions used (Fig. 5). With respect to model complexity we can see that some of the best models are relatively small. This is important since, as we have mentioned, we intend to embed our classification model as JavaScript in the web page. The results for AdaBoost are not shown in the tables because the resulting model is an ensemble of ten base classifiers and hence would be overly complex for our purposes. The results of the PART method do not include model complexity due to WEKA not providing these through its API. We, however, expect PART models to show similar patterns to those of J48 and Ripper in that we would expect the GP and MetaCost to build smaller and more accurate model than the other methods studied in the paper. Random oversampling and SMOTE tend to build more complex models suggesting their susceptibility to overfitting.

```
xrange <= 0.244042 AND yrange <= 0.741614 AND
mouseout <= 10: BROWSE

mousemove <= 12 AND mousemove <= 6: BROWSE

keyup > 1 AND keydown > 3: BROWSE

mousemove > 12 AND click <= 0 AND xrange > 0.304487 AND
focusin <= 0 AND yrange > 0.609193: INT

scroll <= 0 AND click <= 0 AND focusout <= 0 AND
mousemove > 16 AND xrange > 0.258339: INT

scroll > 0 AND yrange <= 0.567992: BROWSE

keyup <= 0 AND mouseout <= 0 AND focusin <= 0: BROWSE

keydown <= 1 AND focus <= 0 AND blur <= 0 AND
mousedown <= 1 AND scroll <= 1 AND yrange > 0.024307: INT

keyup <= 0 AND xrange > 0.221064 AND mousemove > 14 AND
mousedown <= 2 AND focusout <= 0 AND yrange > 0.577115: INT

blur <= 0 AND scroll <= 14: BROWSE

focus <= 0: INT

: BROWSE
```

Figure 5: Rule based model (decision list) for the Olympic data set.

With regard to Precision and Recall, the Olympic data set seems to represent an easier problem than the Soccer data set, perhaps because it is less imbalanced. Precision is in general quite low for the Soccer data (between 0.1 and 0.3) but higher for the Olympic data (0.3 to 0.53). Recall is also higher in general for the Olympic data set. Low values of precision are not very practical for the application we have in mind. An interest model with a precision of 0.2 will fire wrongly 80% of the time. We must note, however, that at deployment time, the models can be used with higher thresholds, which increases precision (but lowers recall). Figures 6 and 7 show the precision and recall curves for the best performing models, MetaCost and AdaBoost. As shown in Figure 6 for all base algorithms AdaBoost has the ability to deliver higher precision models than MetaCost. When using PART as the base algorithm, higher precision models are obtained by MetaCost when high recall is desirable. However, the difference in precision is rather small. Figure 7 shows the corresponding precision-recall curves for the Soccer data set. When using PART or Ripper as the base algorithm, MetaCost can learn models of higher precision when recall is in the range 0.2 to 0.6 in the case of PART and 0.1 to 0.4 in the case of Ripper. However, AdaBoost once again can provide higher precision models in general.

6. CONCLUSION

In this paper we have studied a large number of machine learning techniques for modeling user interest. We have used two data sets and assessed the techniques using a demanding experimental evaluation methodology. All these techniques (with the exception of random oversampling) are adequate, in principle, for dealing with imbalanced classification problems, as is the case in user interest data sets. We conclude that a few techniques can be quite useful for dealing with these data sets. The most interesting compromise in terms of model complexity and AUC is MetaCost using PART as the base classification algorithm. AdaBoost yields higher AUC values but high complexity models. Therefore, our advice for this kind of user interest modeling application would be to use MetaCost and PART unless high precision at the cost of lower recall is desirable in which case it may be worth handling the additional complexity of AdaBoost models built using SMOTE. The accuracy of the AdaBoost based models seems fairly independent of the base classification method used.

Acknowledgment

This work is partially funded by FCT/MEC through PIDDAC and ERDF/ON2 within project NORTE-07-0124-FEDER-000059 and through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-037281.

7. REFERENCES

[1] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms - appendices," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, 2005.

[2] H. Lieberman, "Letizia: An agent that assists web browsing," in *IJCAI (1)*, 1995, pp. 924–929.

[3] M. Claypool, P. Le, M. Waseda, and D. Brown, "Implicit interest indicators," in *IUI*, 2001, pp. 33–40.

[4] J. Goecks and J. W. Shavlik, "Learning users' interests by unobtrusively observing their normal behavior," in *IUI*, 2000, pp. 129–132.

[5] D. Hauger and L. V. Velsen, "Analyzing client-side interactions to determine reading behavior," in *17th Workshop on adaptivity and user modeling in interactive systems (ABIS)*, 2009.

[6] K. Hofmann, C. Reed, and H. Holz, "Unobtrusive data collection for web-based social navigation," in *Workshop on the Social Navigation and Community based Adaptation Technologies*, 2006.

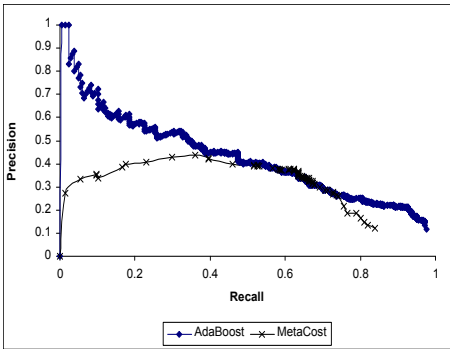
[7] T. F. Syeda-Mahmood and D. B. Ponceleon, "Learning video browsing behavior and its application in the generation of video previews," in *ACM Multimedia*, 2001, pp. 119–128.

Algorithm	Parameters	Precision	Recall	AUC	Model Complexity
J48	No Sampling	0.516±0.031	0.286±0.023	0.74±0.012	67.66±1.38
	RUS 15%	0.446±0.027	0.456±0.022	0.774±0.012	43±3.79
	ROS 1100%	0.339±0.005	0.432±0.01	0.67±0.009	255.38±2.61
	SMOTE 100% N:14	0.42±0.014	0.43±0.016	0.77±0.008	106.88±3.03
	Tomek	0.455±0.016	0.445±0.03	0.74±0.002	75.74±1.34
	GA Wrapper M:10	0.536±0.033	0.259±0.015	0.816±0.005	20.76±1.77
	MetaCost C: 5 M:10	0.378±0.009	0.622±0.014	0.802±0.003	20.28±1.453
	AdaBoost SMOTE 1300% N:13	0.391±0.012	0.493±0.023	0.834±0.008	-
PART	No Sampling	0.423±0.017	0.219±0.013	0.745±0.014	-
	RUS 25%	0.408±0.009	0.37±0.006	0.771±0.009	-
	ROS 100%	0.357±0.022	0.335±0.022	0.714±0.008	-
	SMOTE 1300% N:3	0.365±0.014	0.516±0.006	0.802±0.008	-
	Tomek	0.409±0.008	0.383±0.017	0.768±0.003	-
	GA Wrapper	0.469±0.025	0.222±0.03	0.805±0.009	-
	MetaCost C: 7 M:4	0.306±0.006	0.671±0.014	0.809±0.009	-
	AdaBoost Tomek	0.424±0.012	0.444±0.021	0.831±0.002	-
Ripper	No Sampling	0.443±0.015	0.259±0.014	0.618±0.009	11.56±1.24
	RUS 10%	0.442±0.008	0.413±0.015	0.777±0.001	4.51±0.019
	ROS 500%	0.331±0.01	0.532±0.013	0.717±0.009	144.83±3.9
	SMOTE 900% N:3	0.324±0.006	0.547±0.011	0.792±0.004	94.22±2.06
	Tomek	0.448±0.013	0.423±0.014	0.697±0.006	15.33±0.66
	GA Wrapper	0.317±0.017	0.487±0.033	0.65±0.099	9.025±0.694
	MetaCost C: 8 M:2	0.365±0.007	0.56±0.012	0.742±0.009	32.89±2.826
	AdaBoost RUS 20%	0.413±0.013	0.394±0.013	0.837±0.004	-

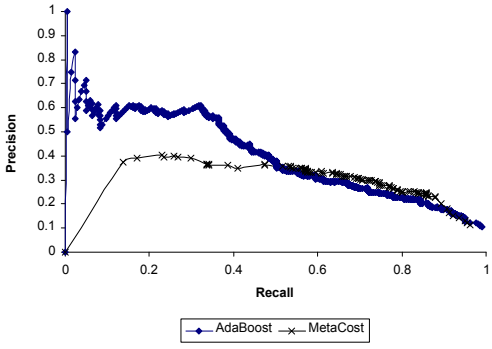
Table 3: Summary of Results for Olympic Data

Algorithm	Parameters	Precision	Recall	AUC	Model Complexity
J48	No Sampling	0.0	0.0	0.527±0.037	1.82±0.55
	RUS 30%	0.025±0.049	0.004±0.008	0.6±0.003	4.88±0.119
	ROS 100%	0.102±0.045	0.111±0.007	0.655±0.016	53.48±2.75
	SMOTE 100% N:7	0.131±0.058	0.15±0.019	0.744±0.015	38.34±1.37
	Tomek	0.05±0.098	0.02±0.004	0.561±0.029	3.72±0.955
	GA Wrapper	0.278±0.073	0.082±0.027	0.776±0.014	13.32±1.12
	MetaCost C:45 M:5	0.068±0.0003	0.915±0.008	0.775±0.006	3.86±0.165
	AdaBoost RUS 65%	0.223±0.028	0.147±0.023	0.799±0.018	-
PART	No Sampling	0.162±0.045	0.058±0.016	0.687±0.02	-
	RUS 80%	0.15±0.015	0.065±0.006	0.73±0.014	-
	ROS 100%	0.227±0.023	0.193±0.025	0.606±0.046	-
	SMOTE 1300% N:5	0.127±0.009	0.308±0.029	0.722±0.032	-
	Tomek	0.145±0.02	0.065±0.006	0.708±0.021	-
	GA Wrapper	0.257±0.065	0.074±0.023	0.729±0.02	-
	MetaCost C: 20 M:5	0.145±0.012	0.584±0.028	0.852±0.01	-
	AdaBoost SMOTE 1300% N:6	0.191±0.016	0.16±0.014	0.807±0.02	-
Ripper	No Sampling	0.297±0.096	0.045±0.018	0.52±0.012	2.9±0.4
	RUS 5%	0.258±0.045	0.074±0.032	0.73±0.032	3.36±0.325
	ROS 1100%	0.1512±0.015	0.247±0.028	0.621±0.013	63.89±1.96
	SMOTE 500% N:14	0.219±0.021	0.35±0.042	0.686±0.013	39.63±1.40
	Tomek	0.214±0.047	0.041±0.018	0.528±0.014	2.94±0.68
	GA Wrapper	0.31±0.046	0.083±0.13	0.544±0.011	4.233±0.462
	MetaCost C: 40 M:5	0.155±0.017	0.27±0.031	0.643±0.01	13.17±0.779
	AdaBoost SMOTE 1100% N:13	0.157±0.008	0.152±0.006	0.806±0.025	-

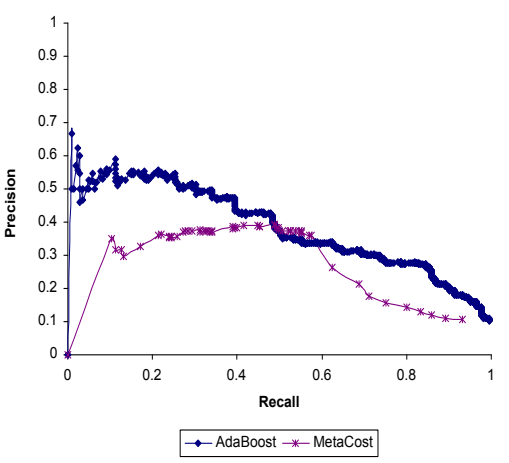
Table 4: Summary of Results for the Soccer Data



(a) J48

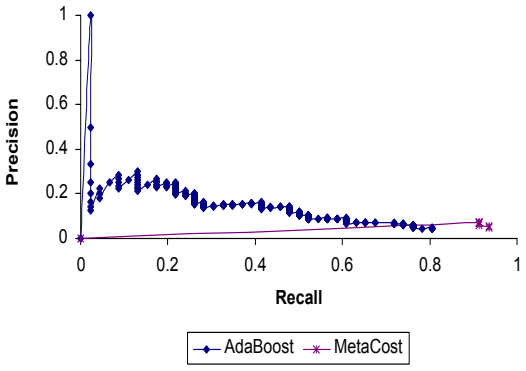


(b) PART

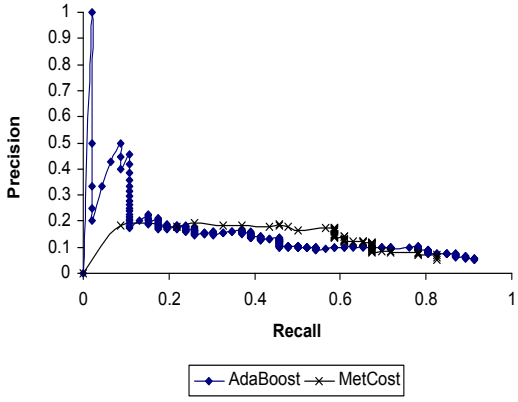


(c) Ripper

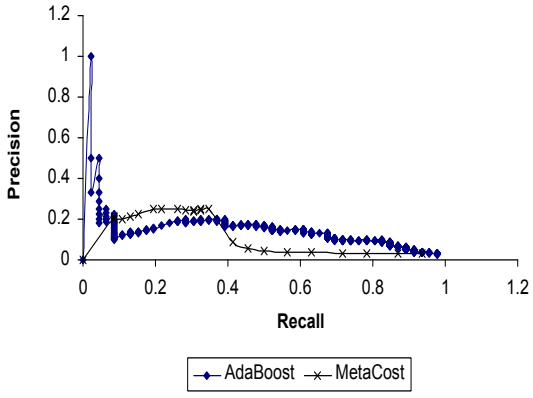
Figure 6: Recall-Precision Curves for select models for the Olympic Data Set



(a) J48



(b) PART



(c) Ripper

Figure 7: Recall-Precision Curves for select models for the Soccer Data Set