PIXS: Programmable Intelligence for Cross-Platform Socialization

Pili Hu
Department of Information Engineering
The Chinese University of Hong Kong
hupili@ie.cuhk.edu.hk

Junbo Li
Institute of Network Technology
Beijing University of Posts and Telecommunications
lijunbo@bupt.edu.cn

Wing Cheong Lau
Department of Information Engineering
The Chinese University of Hong Kong
wclau@ie.cuhk.edu.hk

ABSTRACT

With the proliferation of the emerging Online Social Networks and other conventional communication services, there is an increasing need for a tool which can facilitate individual users to effectively socialize across multiple, heterogeneous platforms. While the diverse nature of the heterogeneous services already makes the design of a cross-platform socialization tool challenging, an even more daunting task is to tackle the "noisy" nature of the Social Networking Services (SNS). Existing solutions all lack flexibility and extensibility, especially in supporting advanced users to better manage their cross-platform socialization via customized information processing. In this paper, we propose PIXS (Programmable Intelligence for Cross-platform Socialization) – an open-source, extensible middleware which provides efficient information acquisition and dissemination across heterogeneous SNSs. A distinguishing feature of PIXS is its support of script-based operations. As a proof-of-concept to demonstrate the flexibility and effectiveness of PIXS, we have developed for it a Python-based semi-supervised learning application which can prioritize incoming messages from different platforms via a Rank Preserving Regression (RPR) framework. This framework can readily incorporate the domain knowledge of the end user. Our SGD-based approach also enables adaptive and incremental training of the ranking system according to the gradual evolution of the user preference. Performance evaluation based on real message traces shows that the proposed system can boost the user's efficiency in identifying and forwarding important messages across heterogeneous SNS platforms. Additional use-cases of PIXS are also discussed.

Categories and Subject Descriptors

H.3.4 [**INFORMATION STORAGE AND RETRIEVAL**]: Systems and Software; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning

Keywords

Social Networking Services; Middleware; Personalization

1. INTRODUCTION

A key function of SNS like Facebook, Twitter and Sina-Weibo, is to support the dissemination of information among a specific group/ community of users. From this viewpoint, many other conventional communication services can also be abstracted using the same primitives of SNS. Consider the following examples:

- In standard SNS, users first specify their relationship with each other (in form of directed or undirected links) and subsequent information is automatically shared according to the topology of the friend-network and/or other additional community structure associated with the friend-networks.
- Under conventional communication services such as those supported by the telephone, SMS and email networks, relationships between users are not declared explicitly or in advance. Instead, the participants of an information-sharing session are defined in an ad hoc and on-demand fashion during "call initiation" by specifying the list of intended recipients. Both uni- and bi-directional communications can be supported.
- For other one-way information distribution channels such as RSS feeds, blogs, and even search engines, user subscribes to the information source by specifying the keyword/ hash-tag or writing a script to monitor the response of the search engine to a particular keyword. The subscription actions implicitly establish the logical topology for information dissemination, which is relatively static and with directed links in nature.

From the above examples, one can see the major differences between the different types of SNS and conventional information services are: 1) whether links/relationships between different parties are declared explicitly beforehand or implicitly established dynamically; 2) whether the information flow is uni- or bi-directional; 3) whether a user is allowed to read and/or write from/into the information sharing channel; 4) whether user authentication and/or authorization are required.

Parametrized according to the above dimensions, heterogeneous communication services can be abstracted and modeled under a single generic template. Such a unified view also plays a critical role in realizing effective information processing/ dissemination across multiple platforms. For example, with the unification of their interfaces, customized user operations, preference feedback as well as programmable intelligence on one platform can be readily transferred and reused on other platforms.

Nowadays, many users already actively communicate through more than one of the information-sharing platforms. It is noteworthy that users often consciously or subconsciously "stitch" the heterogeneous platforms together by selectively relaying messages across different platforms after some manual filtering/ editing. For example, after reading a "juicy" gossip from a blog or subscribed email-list, you may forward it manually to your personal friends on Facebook, but not to your professional colleagues on LinkedIn. In fact, such cross-platform forwarding operations can be viewed as the formation of new multi-modal SNS which overlays on top of the existing SNS as well as other communication services.

With these goals in mind, we have designed and implemented PIXS which enables easy manipulations and information processing across multiple heterogeneous SNS/ communication platforms. Besides the support of easily configurable default forwarding/ filtering policies for its basic users, PIXS allows advanced users to program certain intelligence into the system via simple Python scripts. Instead of targeting the almost-impossible goal of fully-automated content selection, editing and relay, PIXS provides a framework under which explicit user feedbacks can be readily incorporated into the information processing cycle to boost its effectiveness and accuracy. As a proof-of-concept, we have designed and integrated a customizable message-ranking framework which can help a user to prioritize incoming messages and determine their suitability to be forwarded to different communication channels. In short, PIXS is to help users to efficiently identify important/ interesting incoming messages amid large volume of input information streams. The final operations like tagging, editing and forwarding those seemingly important messages remain to be under the manual control of the PIXS user.

The rest of the paper is organized as follows. In Section 2, we briefly survey related work. In Section 3, the system architecture of PIXS is presented. In Section 4, we formulate the Rank Preserving Regression problem for message prioritization. In Section 5, we transform the problem and propose to use Stochastic Gradient Descent (SGD) as the training algorithm. In Section 6, we evaluate all our system in terms of efficiency, accuracy, robustness, and flexibility. Lastly, we conclude this paper and propose future work in Section 7.

2. RELATED WORK

There are many existing aggregation services on the Internet. Some of them focus on dealing with specific type of resources, e.g. Google Reader [7] is for RSS feeds. Others can handle heterogeneous resources while supporting rule-based filtering at different levels of sophistication. For example, IFTTT [13] abstracts Internet-based information services as "channels" and allows users to define "IF-This-happens-Then-do-That" (IFTTT) forwarding recipes. In contrast, Yahoo Pipes [14] supports fewer platforms but has more sophisticated processing by allowing users to design a storyboard of logical operations (e.g. condition, loop, etc) between different resources (e.g. webpage, RSS, etc) and connect them using "pipes". Yoono [8], SocialOomph [9], and Hootsuite [10] are other example services which provide partial interoperability between heterogeneous platforms with some basic personalization functions. While these services may satisfy novice users, they have at least one of the following problems: 1) Only configurable but not programmable, which severely limits their functions; 2) Non open-source and thus less reusable for automation; 3) Only support a small subset of platforms and are not readily extensible to other platforms. There are also other open source projects which aim to bridge different OSNs. For example, OpenSocial [11] abstracts multiple OSNs using the same interface. However, its design is mostly server-side oriented and requires a steep learning curve if users just want some simple programmable customization to his/her own SNS operations. Another example we found during the preparation of this paper is ThinkUp [12]. It is an open-source web service which reads messages from several OSNs and stores them in a local database. Further analysis can be performed on ThinkUp to help the users to find important information. However, ThinkUp does not provide a comprehensive abstraction of heterogeneous SNS like PIXS and ThinkUp's web service nature requires more heavy-weight infrastructure support (e.g. LAMP environment), making it hard to run on mobile devices.

Besides interface heterogeneity, another major challenge of operating multiple SNS is the problem of information explosion even under selective channel subscriptions and careful formation of one's friends-network. As such, additional personalization, e.g. message filtering and prioritization, is in order. Personalization is a research area of its own right even for the case of single-platform support and many service providers have developed their own machine learning algorithms for such purpose. For example, users of SinaWeibo and Facebook can select between the reranked or reverse chronological timeline; GMail users find their messages classified into "important" and "ordinary" types. These existing approaches focus on training with implicit user feedbacks, e.g. how long a user stays on one message. Users are seldom actively involved in the training cycle even if they want. It is asking machine to solve a harder-than-necessary problem. Worse still, the solutions provided by the service providers mostly take the heavy-weight server-side approach targeting the mass while providing little flexibility for advanced users.

3. SYSTEM ARCHITECTURE

In designing PIXS, we have adopted the following principles: 1) Focusing on solving the 80% problems; 2) Combining human and machine intelligence; 3) Staying open to support future service evolution ; 4) Trading execution performance for script development efficiency. Based on these principles, we divide the architecture of PIXS into the following key components:

- PIXS Middleware [1] is the middleware which supports the interfacing with heterogeneous SNSs. Other developers can write "plugins" to enable the support of new platforms. Without any modifications, applications built on PIXS are readily available for any new platforms to be added in the future.

- PIXS Front-End [2] contains a Web UI (WUI) via which an user can check home timeline, update status and

[1]Source code in `https://github.com/hupili/snsapi/`
[2]Source code in `https://github.com/hupili/sns-router/`

tag incoming messages. The user will get prioritized timeline from WUI if ranking module is enabled.

- Rank Preserving Regression (RPR) is the framework for prioritizing the timeline, which can support light-weight, client-side learning via the Stochastic Gradient Descent (SGD) approach. More details about RPR and SGD will be discussed in Section 4 and 5.

Fig. 1 depicts the architecture of PIXS Middleware, which consists of the following three layers: 1) The Interface Layer (IL) where SNSBase is the base class for the SNS. One can derive from it to implement the actual logic required for interfacing with each SNS platform. New platforms are enabled by writing a "plugin". Types defined in IL are JSON- and Pickle- serializable. 2) The Physical Layer (PL) that realizes common operations like HTTP request/response, OAuth, and Error definition. This is to facilitate the development of additional plugins for future platforms. 3) The Application Layer (AL) provides a container class to hold a set of SNSBase instances. With this container class, a user can easily perform batched operations in a cross-platform fashion.

A Command-Line Interface (CLI) in form of a Python shell has also been developed for PIXS. It can interface with other programming languages via standard input/ output streams. By providing this flexible and powerful middleware, we hope to attract contributors from the open-source community to develop additional cross-platform applications and enable more platforms by writing plugins. Another objective is to enable researchers to collect relevant information from heterogeneous SNS in a convenient and unified way. This can in turn speed-up the prototyping of new data-mining/ machine-learning based applications for social networking.

The focus of PIXS Middleware is to enable interfacing with heterogeneous SNS. This is achieved by abstracting common SNS primitives and unifying the data structures for different platforms. It adopts a synchronous model due to its nature and applications have to handle the scheduling themselves. Although the CLI brings much convenience to advanced users, it is still hard for ordinary users to learn the basic operations. Towards this end, we build the PIXS Front-End. It provides an asynchronous queue structure with persistent storage. Upon the queue facility a more user friendly Web UI (WUI) is built using Bottle [15] as the micro-framework. Users can read, tag and relay messages through the WUI. A ranking module (RPR-SGD to be detailed later) is running in the backend to prioritize incoming messages. With PIXS, users can spot, examine and route important messages among multiple platforms more efficiently.

4. PROBLEM FORMULATION

With the PIXS components, users can operate multiple SNS platforms easily. Thanks to the open-source and modular nature of PIXS, users with basic programming knowledge can customize their SNS experience by writing simple Python scripts. Furthermore, we have also built a general algorithmic framework based on this system to support more sophisticated functions such as re-construction of the user's timeline with prioritized messages extracted from multiple information sources. With the help of this framework, users will be more efficient in identifying and spreading high-value messages in a cross-platform fashion. In this section, we formulate this challenge as a Rank Preserving Regression problem.

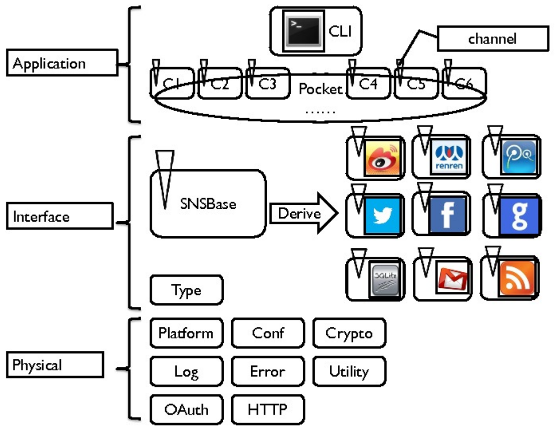

Figure 1: Architecture of PIXS Middleware

Recall that users can tag messages via the WUI of PIXS. The meaning of each tag depends on the users and they can add their own tags. While the relation between tags and messages is in general a many-to-many mapping, we hereby assume it to be one-to-one for simplicity. Furthermore, an imaginary "null" tag is assigned to each message by default after the user marks it as read. Under this setup, every message has one and only one tag.

Suppose we have N messages $m_1, m_2, \ldots, m_N$ and we can extract a K-dimensional feature-vector for each of the messages. By combining the feature-vectors of the N messages, we yield:

$$X^{\mathrm{T}} = [x_1, x_2, \ldots, x_N] \tag{1}$$

where $x_i \in \mathbb{R}^K$ is the feature vector of message m_i. Denote the tag of message i by t_i. Based on this information, we want to rank messages according to their importance perceived by the user.

Towards this end, our first attempt was to formulate the task as a classification problem. To be more specific, we tried to automatically assigning a tag to each message according to the classification outcome and users can choose to read messages with a particular tag-type. We tested the Logit classifier and J48 using Weka [6] with default settings. The test results indicated poor performance of such fine-grained approach. Furthermore, the classification approach also suffers from the following problems: 1) Classifiers usually output rules with rigid cut-off thresholds ; 2) Human can only process messages sequentially so prioritization of different messages is already adequate.

Based on these observations, we proceed to an alternative regression-based formulation. The system will assign a score to a message when it first arrives. This score is stored in database and can be efficiently retrieved to support the necessary operations such as the construction of a ranked timeline for a user.

The next question is how to compute a single score based on the feature values. As a starter, we try the linear combination approach, i.e. let the score $y = Xw$. For one, non-linear combination can be casted to linear combination by extending the dimension of features. Furthermore, the output of linear combination is highly interpretable, e.g. "+10 if the message comes from User X", e.g. "+2 if the message comes from Platform Y", "+5 if the message is about Topic Z", etc. Such an intuitive, easily-understandable approach is particularly desirable as it can encourage users to further customize their timeline by incorporating additional features. Based on this setup, we yield the following regression formulation:

$$\underset{w}{\text{minimize}} \quad ||y - Xw||_2^2 \tag{2}$$

where y_i is the score for message i. If one can find a set of weights w that can result in a good match to y under the training data, we can expect the same set of weights to perform reasonably well on future incoming messages. Under this framework, the processing steps for a newly arrived message include: 1) Extract features from the message ; 2) Use learned w to generate a score, $\hat{y}_i = x_i^{\mathrm{T}} w$; 3) Store $\hat{y}_i$ in the database; 4) Upon query, sort messages by $\hat{y}_i$.

One difficulty of this approach is that y is unknown and it is very hard for users to generate consistent y's even for the training data. Another problem is that as the interest/ taste of a user evolves, he/she has to re-grade all the prior messages and train the weights again, which can be a very tedious process. To work around, we instead ask users to specify their *relative* preference between different pair of messages. This alternate treatment leads to the following new formulation:

$$\underset{y,w}{\text{minimize}} \quad ||y - Xw||_2^2 \tag{3}$$

$$\text{s.t.} \quad y_i > y_j,\ \forall (m_i, m_j) \in E \tag{4}$$

where E is a set of message tuples where (i, j) means message i is preferred than j. Note that y in this formulation is an unknown variable. This makes the problem different from ordinary regression. Furthermore, E only specifies a partial ordering which, to our best knowledge, results in a non-standard regression problem. For now, we refer it as **Rank Preserving Regression (RPR)**. We will describe the detail procedure for the construction of E in Section 5.1.

5. ALGORITHM DESIGN

In this section, we discuss the design of the algorithm for training the RPR model. First, we describe how the preference constraints E are obtained. We then tackle with challenges associated with the resultant optimization problem. Both algorithmic and practical considerations will be covered.

5.1 Inducing Preference Relations on Graph

Fig. 2 illustrates how preference relations of tags are derived. Ovals stand for user defined tags. Solid arrows are user specified preference (extracted from a JSON configuration file). Note that no matter what the meaning of the tags is, the user should be able to tell which tag is preferred. We state $T_u \succeq T_v$ if tag T_u is preferred than T_v. Consider the construction of the tag graph $G_T =< T, P >$, where T is the set of tags and $P = \{(T_u, T_v) | T_u \succeq T_v\}$. If there exists a path from T_x to T_y, we can conclude that $T_x \succeq T_y$. In this way, we extract additional partially ordered tag pairs besides the ones explicitly specified by the user. Since the tag graph is very small in practice, we simply invoke the Floyd algorithm [2] to induce those path preferences instead of deriving on our own. The next step is to

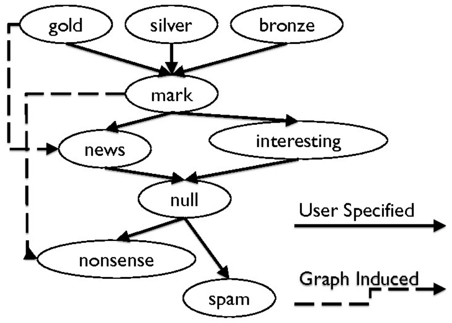

Figure 2: Graph Induction of the Relative Preference Relations of Tags

collect all of the path preferences in one set P_I (named after "Induced Preference"). Notice that $P \subset P_I$ by nature. As such, we can define a message graph $G_M =< M, E >$, where $M = \{m_1, m_2, \ldots, m_N\}$ is the set of all messages and $E = \{(m_i, m_j) | (t_i, t_j) \in P_I\}$ encodes a partial ordering of the messages.

This tag based preference induction has some desirable properties. (1) Flexible: One can either tag in a coarse-grained manner like "good" and "bad", or in a finer-grained manner like "machine learning", "social computing", and "networking". (2) Efficient: The user does not have to give every message a grade in order to let machine learn how to prioritize them. (3) Adaptive: When user's interest shifts from topic to topic or from followee to followee, one only needs to change a subset of edges in the preference configuration.

5.2 Direct Optimization Solver

The RPR formulation is in essence a Quadratic Programming (QP) problem. However, the QP approach has the following drawbacks: (1) Solving QP is costly. It is particularly problematic if one wants to run PIXS on mobile devices ; (2) Typical QP solvers only support batch-mode operations ; In our problem, m_i comes continuously and E is also evolving. If we want to capture time varying user interest, we must solve the QP in an incremental manner. (3) The need of a QP solver will introduce more dependencies to our project, making it less portable. Due to these limitations, we have derived a better training approach by further transforming the QP formulation.

5.3 Problem Transformation

We first convert the inequality preference constraints into equality constraints using indicator function: $I[y_i > y_j] = 1,\ \forall (m_i, m_j) \in E$. Since $I[.]$ only takes 0 or 1, the constraints become: $1 - I[y_i > y_j] \leq 0,\ \forall (m_i, m_j) \in E$.

The Lagrangian of the corresponding problem is:

$$L(y, w, \mu) = ||y - Xw||_2^2 + \sum_{k=1}^{|E|} \mu_k (1 - I[y_i > y_j]) \tag{5}$$

We then lower bound the optimal value of the original problem by solving the Lagrangian dual problem:

$$v_d^* = \sup_{\mu \geq 0} \inf_{y,w} L(y, w, \mu) \tag{6}$$

The point $(\bar{y}, \bar{w}, \bar{\mu})$ which attains the v_d^* can be used to approximate the optimizer for the original problem. $((\bar{y}, \bar{w})$

does not have to be the optimizer for original problem). The physical meaning of $L(y, w, \mu)$ is also clear: The first term penalizes unmatched predictions with respect to observations; The second term penalizes unsatisfied relative preference relations (μ_k is non-negative).

Even with this modification, directly solving the Lagrangian dual problem remains difficult. We therefore restrict ourselves to a family of (y, w, μ) (instead of exploring the whole space of (y, w, μ)) where:

$$\begin{aligned} y &= Xw && (7) \\ \mu_i &= \lambda \geq 0, \forall i = 1, 2, \ldots, |E| && (8) \end{aligned}$$

By doing so, it implies that: 1) we only consider the solutions which provide a perfect match between all predictions and observations; 2) we treat all relative preference relations equally important.

With this setup, we have $\tilde{L}(y, w, \mu) = \lambda \sum_{k=1}^{|E|}(1 - I[y_i > y_j])$ and the supinf problem of Eq. 6 becomes the following optimization:

$$\begin{aligned} \underset{y,w}{\text{minimize}} \quad & \sum_{(m_i,m_j)\in E} 1 - I[y_i > y_j] && (9) \\ \text{s.t.} \quad & y = Xw && (10) \end{aligned}$$

Note that we can substitute y with Xw in the objective, to yield an unconstrained optimization with respect to variable w. This formulation also reduces the number of variables from $(N + K)$ to K, which is significant for the practical problem on hand.

5.4 Stochastic Gradient Descent

We can leverage first-order optimization techniques to solve the newly formulated problem in Eq. 10. First, we approximate indicator using Sigmoid function, $S(x) = \frac{1}{1+e^{-\beta x}}$, where β is a scaling factor to control the approximation rate. We then define

$$f(w) \equiv \sum_{(i,j)\in E} 1 - S(y_i - y_j) \quad (11)$$

where $y_i = (Xw)_i$ and $(i, j) \in E$ is a shorthand notation for $(m_i, m_j) \in E$. The gradient of $f(w)$ is given by:

$$\begin{aligned} \nabla f(w) &= \sum_{(i,j)\in E} \nabla f_{ij}(w) && (12) \\ \nabla f_{ij}(w) &= \beta(1 - S(y_i - y_j))S(y_i - y_j)(x_j - x_i) && (13) \end{aligned}$$

where $\nabla f_{ij}(w)$ is the "stochastic gradient" [1] with respect to the pair of relative preference relation (i, j). We randomly loop around all the induced pair of relations and descend $f(w)$ along the direction opposing to the stochastic gradient.

6. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed RPR-SGD scheme. We first introduce the data set and propose to use the Kendall's tau correlation coefficient as a performance metric. Interested readers can refer to [5] for a demo of adding extra features, which shows the flexibility of RPR-SGD.

6.1 Data Set

We collected real message trace from an instance of PIXS over a 1-month period. Table 1 summarizes some basic statistics. We then derive training and testing relative preference relations as follows: 1) Select all tagged messages (a total of 900); 2) Sample an equal number of untagged messages (about 900) and assign the "null" tags to them; 3) Randomly partition the candidate messages (about 1.8K) into equal-sized training set (M_{train}) and testing set (M_{test}); 4) Use graph induction to form $G_{M_{\text{train}}}$ and $G_{M_{\text{test}}}$ which yields more than 200K relative preference constraints.

Table 1: Basic Statistics of the Data Set

Item	Value
# of total messages	32533
# of seen messages	7553
# of tagged messages	924
# of forwarded messages	167
# of derived pairs (training)	231540
# of derived pairs (testing)	229009

6.2 Evaluation Criterion

Kendall's tau correlation coefficient [3] is a good measure of ranking. The modified version for our problem is defined as (and referred to as the Kendall's score for short):

$$K = \frac{\sum_{(i,j)\in E_{\text{test}}} I[\hat{y}_i > \hat{y}_j] - \sum_{(i,j)\in E_{\text{test}}} I[\hat{y}_j > \hat{y}_i]}{|E_{\text{test}}|} \quad (14)$$

where $\hat{y}_i$ is the predicted score of message i using the learned weights w and E_{test} is the set of edges in $G_{M_{\text{test}}}$. Kendall's tau correlation computes difference between satisfied and unsatisfied pairs and divide it by the total number of relations. K is in the range $[-1, 1]$ and the larger the better. When $\hat{y}_i$'s are assigned randomly, $K = 0$.

6.3 Sample Features

To emphasize the easy-to-use nature of our RPR-SGD framework, we restrict ourselves to readily-extractable message features including: Whether or not the message contains a link; Whether this message comes from the PIXS user; Full message length (including original post and repost); Cleaned message length (i.e. without "@xxx", etc); Original message length; Message topics; User topics. Most of the features are self-explanatory. We only elaborate the last two features. To get topic types, we manually classify the tags defined by the user into several topics, e.g. "tech", "news", "nonsense", etc. We use the TF-IDF [4] approach to mine the topics by treating new messages as queries and topics as documents. In this way, we invoke standard Information Retrieval methods to determine the message topics. Also note that different users tend to post different topics. In this case, we want to analyze the topics of a user. The only difference to mining the message topic is that we treat users as terms in TF-IDF. We remark that only rudimentary topic mining algorithm is being used here; Further optimization is possible with more advanced topic-mining schemes.

6.4 Performance and Complexity

We have implemented SGD using Python without any code level optimization. Table 2 shows the performance of the resultant trained model under different training time and sizes. In this evaluation, step size is chosen as $\alpha = 10^{-2}$. We see that the value of K for the testing set becomes larger than 0.7 after several dozens of thousand of iterations, which means 85% pairs are in correct order (i.e.

$K = (0.85|E| - 0.15|E|)/|E|)$. This is a substantial improvement over the unranked timeline. We have also implemented normal Gradient Descent (GD) as a baseline, which costs about 30sec for each iteration and requires 15 iterations to result in $K \approx 0.75$. This clearly shows the effectiveness of SGD. It is important to note that SGD can be executed in the backend constantly with new incoming data and users can query the current best ranking at any time. In this way, PIXS do not have to block a full training cycle while users are served.

Table 2: Training with SGD

Item	1.	2.	3.
# of rounds	200,000	400,000	1,000,000
wall clock time	32.63s	60.81s	159.57s
K (training)	0.8178	0.8349	0.8414
K (testing)	0.7598	0.7758	0.7865

6.5 Improved User Efficiency

We present the user feedback in Table 3 to show that RPR-SGD improves user efficiency significantly. Here, the time period is divided into weeks to absorb natural variance due to workday vs. weekend or day vs. night. This deployment connects to multiple platforms including Twitter, Renren, SinaWeibo, TencentWeibo, SQLite, Gmail, and some RSS feeds. "A" is the total number of messages fetched by PIXS during the experiment period. "S" is the total number of messages seen by the user. "V" is the number of valuable messages to the user. As a rough estimation, we treat tags that are superior to "null" in the preference graph as valuable ones. The last column "V/S" is the ratio between the number of "Valuable" messages and total messages "Seen".

Table 3: User Efficiency Evaluation

No.	Period	A	S	V	V/S
1	Nov 13 - Nov 20	6578	1489	114	7.6561%
2	Nov 20 - Nov 27	9040	1544	138	8.9378%
3	Nov 27 - Dec 04	8472	846	184	21.749%
4	Dec 04 - Dec 11	8243	225	56	24.889%

During initial deployment, the ranking module was disabled so that we could bootstrap the system with some user data. We subsequently enabled ranking on Nov 26. Table 3 shows that user efficiency is improved to about three times even if we only consider the process that people filter out useful information. Considering the multi-interfacing nature of PIXS, users act much more efficiently in acquiring and disseminating valuable information. Refer to [5] for more details of this experiment.

6.6 Reaction to Noisy Features

In this experiment, we intentionally inject a noisy feature and see how RPR-SGD reacts. We start with features discussed above without "noise", and run enough steps of SGD to yield $K > 0.8$. The absolute values of trained weights for the remaining features are all less than 10. We then inject a noise feature, which is a r.v. $\sim U[0, 1]$, and its weight is initialized to 10. Table 4 shows the training result. Observe that the Kendall's score is close to 0 initially due to the large initial weight we assigned to the noise feature. This corresponds to essentially random ordering of messages. After running more steps of SGD, the magnitude of the weight of the noise feature gradually reduces and the Kendall's score improves accordingly.

Table 4: Robustness Test

	Init	Round=200K	Round=400K
Kendall	0.0772	0.5435	0.8060
w(noise)	10.0	1.3407	-0.0132

7. CONCLUSIONS AND FUTURE WORKS

In this paper, we have described PIXS and demonstrated its support of programmable intelligence which enables users to better manage socialization across heterogeneous SNS. PIXS can be used to establish a new multi-modal SNS which can leverage the communication/ information-storage infrastructure provided by the existing SNS. Besides, it can also help to "unlock" the content originated/ stored under the existing, compartmentalized platforms. Most importantly, the overlaying SNS built by PIXS would allow users to gradually regain control of their personal content and communication archives by automatically forwarding information from those proprietary, closed platforms to other more open, non-commercial systems, e.g. the decentralized online social networking services supported by Diaspora [16]. The middleware can also provide an adaptation layer to web services like ThinkUp [12] to enable more platforms.

Acknowledgements

This research is supported in part by the CUHK MobiTeC fund.

8. REFERENCES

[1] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17:9, 1998.

[2] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.

[3] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[4] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 26(3):13, 2008.

[5] Pili Hu, 2012, SNSRouter – A Framework for Intelligent Message Routing on Heterogeneous SNS, `https://github.com/hupili/sns-router/blob/paper/doc/paper/snsrouter.pdf`

[6] Weka, 2013, `http://www.cs.waikato.ac.nz/ml/weka/`

[7] Google Reader, `http://www.google.com/reader/`

[8] Yoono, `http://www.yoono.com`

[9] SocialOomph, `https://www.socialoomph.com`

[10] Hootsuite, `http://hootsuite.com`

[11] OpenSocial, `http://opensocial.org`

[12] ThinkUp, `https://www.thinkup.com`

[13] IFTTT, `https://ifttt.com`

[14] Yahoo Pipes, `http://pipes.yahoo.com/pipes/`

[15] Bottle, `http://bottlepy.org/docs/dev/`

[16] Diaspora, `https://joindiaspora.com`