



# AN ABSTRACT OF THE DISSERTATION OF

Mohammad S. Sorower for the degree of Doctor of Philosophy in Computer Science  
presented on November 30, 2015.

Title: Improving Automated Email Tagging with Implicit Feedback

Abstract approved: \_\_\_\_\_

Thomas G. Dietterich

Machine learning systems are generally trained offline using ground truth data that has been labeled by experts. However, these batch training methods are not a good fit for many applications, especially in the cases where complete ground truth data is not available for offline training. In addition, batch methods do not perform well in applications where the learning system is expected to quickly adapt to changes in the data with a non-stationary distribution and also remain resistant to label noise. Online learning algorithms provide solutions to these challenges, but these algorithms often assume that the ground truth is available after making every prediction.

In this thesis, we describe the ‘online email tagging’ problem where an underlying algorithm predicts a set of user-defined tags for an incoming email message. The email client user interface displays the predicted tags for the message, and the user doesn’t need to do anything unless those predictions are wrong (in which case, the user can delete the incorrect tags and add the missing tags). This means that the learning algorithm never receives confirmation that its predictions are correct – it only receives feedback when it makes a mistake. This violates the assumption of most online learning algorithms, and can lead to slower and less effective learning. In many cases, the learning algorithm would benefit from positive feedback, i.e., confirmation of correct predictions.

One could assume that if the user never changes any tag, then the predictions are correct. But users sometimes forget to correct the tags, presumably because they are focused on the content of the email messages and fail to notice incorrect and missing tags. The aim of this thesis is to determine whether implicit feedback can provide useful

additional training examples to the email prediction subsystem of TaskTracer, known as TAPE (Tag Assistant for Productive Email). Our hypothesis is that, the more time a user spends working on an email message, the more likely it is that the user will notice tag errors and correct them. If, after the user has spent enough time working on an email message, no corrections have been made, then perhaps it is safe for the learning system to treat the predicted tags as being correct and train accordingly. We propose four algorithms (and three baselines) for incorporating implicit feedback into the TAPE email tag predictor. These algorithms are then evaluated using (i) email interaction and tag correction events collected from 14 user-study participants as they performed email-directed tasks while using TAPE, and (ii) case studies on real knowledge workers using TAPE to manage their own email messages. The results show that implicit feedback produces important increases in training feedback, and therefore, significantly reduces subsequent prediction errors despite the fact that implicit feedback is not perfect. We conclude that implicit feedback mechanisms can provide a useful performance boost for online email tagging systems. Finally, we perform a simulation study to show how tags could provide services to help with information re-finding and several common tasks that the users often need to perform within the email system. Our simulation results show that tag services have potential to greatly reduce the number of clicks required to perform these tasks.

©Copyright by Mohammad S. Sorower  
November 30, 2015  
All Rights Reserved

# Improving Automated Email Tagging with Implicit Feedback

by

Mohammad S. Sorower

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of  
the requirements for the  
degree of

Doctor of Philosophy

Presented November 30, 2015

Commencement June 2016

Doctor of Philosophy dissertation of Mohammad S. Sorower presented on  
November 30, 2015.

APPROVED:

---

Major Professor, representing Computer Science

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

---

Mohammad S. Sorower, Author

## ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Thomas G. Dietterich, for his precious guidance, supervision, and support throughout the course of my research. I am immensely grateful to him for his patience and moral support at difficult times of the research and the writing of this dissertation. He has constantly guided me in all stages of my research, right from the beginning to the end. I could not have imagined having a better advisor and mentor for my PhD study.

I would like to thank the rest of my committee members: Prof. Prasad Tadepalli, Prof. Alan Fern, Prof. Xiaoli Fern and Prof. Harry Yeh, for their encouragement, and feedback on my research as well as on my dissertation. It is an honor to have such a distinguished group of scientists and researchers in my committee.

I wish to extend special thanks to Michael Slater, who is not only the primary developer of the software (TAPE) used in this research but also is a great researcher. Michael spent innumerable hours developing and debugging the TAPE system. I had frequent deep discussions and collaborations with Michael that contributed significantly to my research. I also wish to thank Janardhan Rao Doppa, Jed Irvine, Javad Azimi, Shubhomoym Das, Liping Liu, Majid Alkaee Taleghan, Tadesse Zemicheal and all my lab mates – discussing with them has always been a source of valuable suggestions.

I am indebted to my many friends and colleagues for their companionship and help during all these years in graduate school and my stay in Corvallis, Oregon. Working with such bright and thriving minds made my doctoral study at Oregon State enjoyable, eventful and memorable.

I gratefully thank and acknowledge the funding agencies that made this dissertation possible. My PhD career and this work was funded by a gift from Intel Corporation to support the TaskTracer project, by DARPA Contract No. FA8750-09-C-0179, and also by DARPA Contract No. 2014-1451 (CFDA 12.XXX).

I owe my deepest gratitude to my parents. My mother has shown immense patience, and have sacrificed so much in order for me to successfully complete my PhD. I sincerely acknowledge that without her enormous sacrifice, patience, encouragement, and support, this dissertation would not have been possible.

Last but not the least, I am filled with appreciation for my dear wife, Mafruhatul

Jannat (Medha), who endured all the stress with me, while undergoing and successfully achieving her own doctorate degree. I am grateful for her patience, encouragement, and love – I am blessed to have her as my wife.



# TABLE OF CONTENTS

	<u>Page</u>
1 Introduction	1
1.1 Thesis Contributions . . . . .	4
1.2 Thesis Organization . . . . .	4
2 Background and Related Work	6
2.1 Tagging of Email Messages . . . . .	6
2.2 Learning from Implicit Feedback . . . . .	7
3 TAPE Implicit Feedback System	12
3.1 TAPE Email Predictor . . . . .	12
3.2 User Interface Instrumentation . . . . .	15
3.3 Baseline Algorithms . . . . .	15
3.3.1 No Implicit Feedback . . . . .	16
3.3.2 Self Training . . . . .	16
3.3.3 Online Learning . . . . .	19
3.4 Implicit Feedback Algorithms . . . . .	20
3.4.1 Simple Implicit Feedback . . . . .	20
3.4.2 Implicit Feedback without SIF . . . . .	20
3.4.3 Implicit Feedback with SIF . . . . .	24
3.4.4 Implicit Feedback with SIF using Learned Weights . . . . .	24
4 The Lab-controlled User Study	27
4.1 Dataset of Tagged Email Messages . . . . .	27
4.2 The User Study . . . . .	28
4.3 Post-study Simulation . . . . .	30
4.4 Results Analysis . . . . .	33
5 Knowledge Worker Case Study	45
5.1 Case Study 1: A Graduate Student . . . . .	45
5.1.1 The Data Set . . . . .	45
5.1.2 Parameter Learning . . . . .	46
5.1.3 Results Analysis . . . . .	49
5.2 Case Study 2: A Professor . . . . .	59
5.2.1 The Data Set . . . . .	59

## TABLE OF CONTENTS (Continued)

	<u>Page</u>
5.2.2 Parameter Learning . . . . .	60
5.2.3 Results Analysis . . . . .	65
5.3 Summary . . . . .	71
6 Tag-based Email Services . . . . .	74
6.1 Email Services through TAPE . . . . .	75
6.2 Click Cost and Simulation of Tag Services . . . . .	75
6.3 Simulation Results . . . . .	77
6.4 Summary . . . . .	84
7 Conclusion and Future Work . . . . .	85
7.1 Future Work . . . . .	86
Bibliography . . . . .	87
Appendices . . . . .	92
A Subject Recruitment Flyer for the User Study . . . . .	93
B Subject Eligibility Questionnaire for the User Study . . . . .	94
C Sample Email Messages from the User Study . . . . .	96

## LIST OF FIGURES

Figure	Page
1.1 Survival curve (exponential fit to the data) showing the fraction of the messages with bad tags that survived $k$ interactions with the user. . . . .	2
2.1 TAPE online tagging performance when the system is trained on user corrections only and when trained on ground truth tags for every message.	8
3.1 TAPE Email Predictor Tag Interface on Microsoft Outlook. . . . .	13
4.1 Conditional probability distribution, $P(EF   totalIF)$ . . . . .	31
4.2 Implicit feedback captured during the study sessions of one participant. The first session ends after message 66, and the second session ends after message 168. . . . .	34
4.3 Total number of implicit feedback events captured (log scale) for each type of implicit feedback event. ‘Message R/O’ indicates the total number of times a message was opened in Outlook Explorer or in Outlook Inspector.	35
4.4 Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold (TargetEF = 0.20). . . . .	36
4.5 A comparison of the cumulative mistakes of each of the six IF algorithms on the last 70 email messages for six values of TargetEF. . . . .	38
4.6 Total number of training examples for the entire experiment (left axis) and total number of prediction mistakes on the last 70 messages (right axis) for different levels of TargetEF. . . . .	41
4.7 Percentage of training messages correctly confirmed by IF for different levels of TargetEF. . . . .	42
4.8 Total mistakes for different levels of TargetEF. $p$ -value $< 0.05$ for a two-sided Welch’s two sample $t$ -test suggests that we have sufficient evidence to conclude that the total number of mistakes in NoIF is greater than the number of mistakes in IFwSIF. . . . .	43
5.1 Distribution of Tags in Sorower-Data. . . . .	47

## LIST OF FIGURES (Continued)

<u>Figure</u>	<u>Page</u>	
5.2	Implicit feedback events captured in Sorower-Data plotted on a log 10 scale. ‘Message R/O’ indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector. . . . .	48
5.3	Total mistakes made by self-training on Sorower-Data, plotted as a function of ‘Positive Prediction Training Lower Threshold’ and ‘Negative Prediction Training Upper Threshold’. . . . .	50
5.4	Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold, computed on Sorower-Data. . . . .	51
5.5	Weights of the implicit feedback events learned on Sorower-Data. . . . .	52
5.6	A comparison of the cumulative mistakes of each of the seven IF algorithms on Sorower-Data. . . . .	53
5.7	Total number of positive training examples on Sorower-Data. . . . .	55
5.8	Total number of negative training examples on Sorower-Data. . . . .	56
5.9	Total number of training examples (left axis) and total number of prediction mistakes on Sorower-Data. . . . .	57
5.10	Total additional training examples created by IFwSIF and STrain on Sorower-Data. . . . .	58
5.11	Distribution of Tags in TGD-Data. . . . .	59
5.12	Implicit feedback events captured in TGD-Data plotted in log 10 scale. ‘Message R/O’ indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector. . . . .	61
5.13	Total mistakes made by self-training on TGD-Data, plotted as a function of ‘Positive Prediction Training Lower Threshold’ and ‘Negative Prediction Training Upper Threshold’. . . . .	62
5.14	Total mistakes made by self-training on TGD-Data plotted as a bar plot (left axis) to show the effects ‘Positive Prediction Training Lower Threshold’ (right axis) and ‘Negative Prediction Training Upper Threshold’ (horizontal axis). . . . .	63

## LIST OF FIGURES (Continued)

<u>Figure</u>	<u>Page</u>
5.15 Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold, computed on TGD-Data. . . . .	64
5.16 A comparison of the cumulative mistakes of each of the seven IF algorithms on TGD-Data. . . . .	66
5.17 Total number of positive training examples on TGD-Data. . . . .	67
5.18 Total number of negative training examples on TGD-Data. . . . .	68
5.19 Total number of training examples (left axis) and total number of prediction mistakes on TGD-Data. . . . .	69
5.20 Total additional training examples created by IFwSIF and STrain on TGD-Data. . . . .	70
6.1 Expected click cost without and with tag services computed for the 270 message in the user-study for $\eta = 0.0$ and $\gamma = 0.0$ . . . . .	79
6.2 Expected click cost without and with tag services computed for the 270 message in the user-study for $\eta = 0.5$ and $\gamma = 0.5$ . . . . .	81
6.3 Expected click cost without and with tag services computed for the 270 message in the user-study for $\eta = 0.8$ and $\gamma = 0.7$ . . . . .	82
6.4 Expected click cost without and with tag services computed for the 270 message in the user-study for $\eta = 1.0$ and $\gamma = 1.0$ . . . . .	83

## LIST OF TABLES

<u>Table</u>		<u>Page</u>
2.1	Properties of the email system, and the requirements for good email tagging.	7
4.1	The distribution of tags in the email dataset. For each tag, this table shows the percentage of messages that were assigned that tag. This totals to more than 100% because a message may have multiple tags. . . . .	29
4.2	Percentage tag prediction mistakes. . . . .	40
5.1	Summary of Data sets and the Results from the Experiments. . . . .	72

## LIST OF ALGORITHMS

<u>Algorithm</u>	<u>Page</u>
1 NoIF . . . . .	17
2 STrain . . . . .	18
3 Online . . . . .	19
4 SIF . . . . .	21
5 IFwoSIF . . . . .	22
6 IFwSIF . . . . .	23
7 SampleEF( $p, tEF$ ) . . . . .	32

## LIST OF APPENDIX FIGURES

<u>Figure</u>	<u>Page</u>
A.1 A snapshot of the flyer used to recruit subjects for the user study. . . . .	93



## Chapter 1: Introduction

Email was originally designed as a communications application. Over time, it has become a *habitat* for collaboration, a tool for time and task management, a medium for conversations and file transmission, and a mechanism for managing professional and social contacts for knowledge workers [14, 18, 40]. Workflows in organizations are often initiated, discussed, managed, and concluded via email exchanges [27]. *Whittaker et al.* [40] call this *email overload*, and show how the email inbox is employed as a repository for task to-do, to-read, and for tasks or correspondence in progress. However, with the increased volume of email that people receive every day, congested, unstructured and overloaded email boxes with disordered and unprioritized emails pose a critical bottleneck on efficient knowledge work. Tools are needed that can help the user efficiently organize and manage email, so that knowledge workers can keep pace with the stream of tasks. One of the most common practices among email users is to organize email messages manually into folders. This manual process clearly is not efficient and exhibits the same problems as folder-based file systems. There is no easy way to manage cases where an email message is related to multiple tasks or projects. Most email clients support hand-coded rules for tagging or foldering, but these must be manually crafted and they are mostly inflexible [10]. Therefore, recent attention has been drawn to applying machine learning methods to automatically classify email messages into folders or attach appropriate tags to the messages.

The TAPE Email Predictor [36] incorporates automated email prediction into Microsoft Outlook using a multi-label classifier based on the Confidence Weighted (CW) linear classifier. The user interface, which we will describe below, displays the predicted tags on each incoming message. The nice aspect of this is that if the tags are correct, the user doesn't need to take any action. In our experience, the classifier is 80-90% correct, and hence the cost to the user of manually tagging email messages is reduced by 80-90%. If the tags are incorrect, then the user can easily delete incorrect tags and add the correct tags, and this provides training examples to the classifier. However, the classifier receives no feedback when the predicted tags are correct. If the classifier was highly confident of

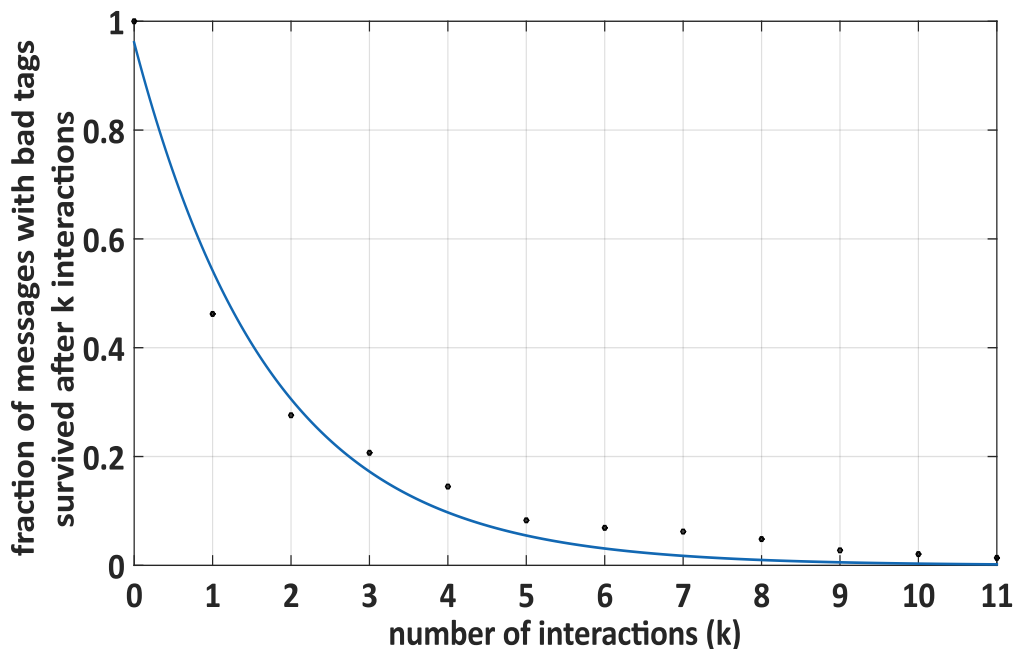


Figure 1.1: Survival curve (exponential fit to the data) showing the fraction of the messages with bad tags that survived  $k$  interactions with the user.

those predictions, then this would not be a problem. (Indeed, the confidence weighted classifier ignores positive feedback on confident predictions.) But if the classifier is not confident, then positive feedback confirming the correctness of those predictions would be very helpful.

The obvious response to this quandry is to employ what is known as “self training”. The classifier could simply assume that its predictions are correct if the user doesn’t make any corrections. However, our personal experience as users of TAPE has shown that once the classifier becomes reasonably accurate, users occasionally fail to provide corrective feedback, especially if they are working quickly or are deeply engaged in a task. Under such conditions, self training is risky.

This motivates us to explore *implicit feedback*. Our hypothesis is that the more time a user spends working on an email message, the more likely it is that the user will notice tag errors and correct them. The survival curve shown in figure 1.1 is plotted using the

data from a lab-controlled user study (described in Chapter 4), and this supports our hypothesis. The horizontal axis shows the number of times ( $k$ ) a user interacted with a message that had an incorrectly-predicted tag, and the vertical axis shows what fraction of the messages (with bad tags) survived  $k$  interactions. The survival curve shows an initial drop for  $k = 1$  and  $k = 2$  followed by slower decrease thereafter. This suggests that most of the bad tags are corrected almost immediately within a few interactions. Therefore, if the user spends a long time reading a message, saves an attachment, forwards the message to someone else, and then replies to the message—all without changing any of the tags—then it is likely the tags are correct. In contrast, if the user only briefly looks at the email message and then moves on to the next message, the tags could very well be wrong and the user failed to notice. In this thesis, we study implicit feedback mechanisms that record events in which the user interacts with an email message. If the total number of these events exceeds a threshold without any tag changes, then the tags are assumed to be correct and the classifier trains on that message.

One might think that this problem is an instance of learning in the bandit framework. In multiclass classification learning under the bandit framework, the learning algorithm only receives feedback that the predicted class is right or wrong. If the predicted class label is wrong, the algorithm is not told what the correct label should have been [8]. In contrast, for the email tagging problem as described above, implicit feedback can either infer that a predicted tag is correct (if the total number of interaction events exceeds the threshold) or it can not draw any inference (if the total number of events is below the threshold). Implicit feedback in this case cannot infer that a predicted tag is wrong. Hence, some of the email messages may not receive any kind of feedback at all. Implicit feedback is also inherently noisy, and therefore, the inferred feedback is not guaranteed to be correct as in bandit framework.

In this thesis, we instrument TAPE to collect events relevant to producing implicit feedback. We propose four algorithms for generating implicit feedback in response to these events as well as three baseline methods. We conducted a lab-controlled user study that collected events in which users are interacting with TAPE-tagged email messages while they carry out various tasks specified by those messages. To evaluate and compare the various implicit feedback algorithms, we reprocess the data from the user study to create simulated users who notice and repair incorrect tags with a specified target probability. We also conducted case studies with real knowledge workers using the TAPE system in

their own email. The results show that by training on tags confirmed by implicit feedback, we can significantly improve the performance of the TAPE email tagging system.

One of the goals of this thesis is to make tags generally useful for supporting user workflows. The most common use-cases where tags have been often exploited are search and retrieval. However, tags can also contextualize the tagged object and user activities [3]. For example, in the case of image tagging, users often label the content of a photograph by entering the names of the people who appear in the photograph, or the location where it was taken. In the case of a tag-based desktop system, the presence of some tag(s) on a document could indicate the user work-context (e.g., project/task), and, therefore, the system can proactively suggest relevant resources for the current context (e.g., recently used documents related to the tag). In this thesis, we utilize the data from the lab-controlled user study to perform a preliminary simulation study to show how tags could provide services to help with information re-finding and several common tasks that the users often need to perform within the email system. Our simulation study shows that tag services have potential to greatly reduce the number of clicks required to perform several common tasks on an email message.

## 1.1 Thesis Contributions

1. Developed an approach to model implicit feedback for a multi-label online email classification problem.
2. Conducted a lab-controlled user study to evaluate the system and to assess the benefits of the implicit feedback models against the baseline algorithms.
3. Conducted case studies with real knowledge workers to evaluate the benefits of the implicit feedback-based models in their everyday usage of emails.
4. Performed a simulation study to assess the potential effectiveness of tag-based access to recent documents, folders, and email addresses.

## 1.2 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews related work in the area of online tagging of email messages and in the area of modeling and applications of implicit

feedback. Chapter 3 describes the TAPE system and its user interface. We then describe the instrumentation that we added to TAPE to collect events relevant to producing implicit feedback. We also describe four algorithms for generating implicit feedback in response to these events as well as three baseline methods. Chapter 4 presents a user study that collected events in which users interacted with TAPE-tagged email messages while they carried out various tasks specified by those messages. We then present and discuss the results of this algorithm comparison. Chapter 5 presents case studies with real knowledge workers using the TAPE system on their own email. It then discusses the results of the above-mentioned algorithm comparison. Chapter 6 describes a simulation study that shows how tags can provide useful services to the user. Finally, Chapter 7 concludes and outlines possible directions for future work.

## Chapter 2: Background and Related Work

### 2.1 Tagging of Email Messages

Email is one of the most efficient and popular means of communication of the late 20th and early 21st century, and it has become an integral part of the personal and professional life of millions of people [24, 39]. For knowledge workers, email is the center for task management, a medium for document delivery and archive, and a cheap but fast way of delivering messages. As described above, a tool that could efficiently organize and manage email messages could be a tremendous value to the knowledge workers.

A substantial body of research has explored text-classification-based batch strategies for email classification [7, 33, 22, 4]. However, in the context of deployed email systems, batch methods are not appropriate. Email readers and messages have specific characteristics that define the expected properties of an email tagging system (Table 2.1). A good email tagging system should take the form of an online, multi-label classifier, and the classifier should be able to adapt quickly in response to user feedback. Furthermore, the classifier should be able to incorporate changes in the tag space and in the distribution of words in the email messages.

A few existing studies have explored online learning methods for email. *SwiftFile* [35] presented an incremental learning algorithm that predicts the three most likely destination folders for an incoming message. This incremental learning method performed better than a periodic learning system (e.g., the classifier is updated after every thirty messages or overnight), because the periodic learning system fails to promptly respond to user corrections. The system starts learning very quickly with only a few messages. However, the user may not want to move some of the email messages into folders. *Krzywicki et al.* dynamically learn a threshold to infer whether or not the user would want to move an incoming message into a folder [25]. An empirical evaluation of six different online learners for email classification was reported by *Keiser et al.* [21]. These online classifiers, namely, Bernoulli Naïve Bayes, Multinomial Naïve Bayes, Transformed Weight-Normalized Complete Naïve Bayes, Term Frequency-Inverse Document Frequency Counts, Online Passive

Aggressive, and Confidence Weighted, were tested using real email collected and tagged using the TaskTracer system [12]. The confidence weighted linear classifier [13] consistently performed better than all of the others.

Table 2.1: Properties of the email system, and the requirements for good email tagging.

Factor	Email System Property	Good Email Tagging
Message-to-tag relationship	Many-to-many	Multi-label classification
Dynamic tag-space	User may create new tags, cease using old tags, or even split a tag into multiple tags	Classifier adapts to tag-space changes
Message structure	Message contains sender, recipients, email-thread, mailing-lists, message subject and body	Classifier exploits these as features
Distribution of features	Non-stationary <sup>1</sup>	Classifier adapts to feature distribution changes
User-feedback/corrections	User provides feedback in response to prediction error and provides no feedback on correct predictions	Classifier learns from incomplete feedback from the user
User-mistakes and mind-changes	User can add/remove tags by mistake, and can change mind about the tags on a message	Classifier can recover from user mistakes and mind changes
User-expectation	User expects the system to make fast predictions and to accept immediate feedback	Online learning algorithm

## 2.2 Learning from Implicit Feedback

The online email tagger trains on user corrections and gradually starts making correct predictions. This significantly reduces (often completely diminishes) the requirement for the user tag-corrections. While this is the objective of such a system, this also means that the classifier gradually receives less and less training. However, as mentioned in Table 2.1, in the case of email messages, it is common for the user to revise the definition of a tag,

<sup>1</sup>for example, if the tag is about a class, then at the start of the term, words like “syllabus” and “registration” are important; at the end of the term words like “exam” and “grades” are relevant.

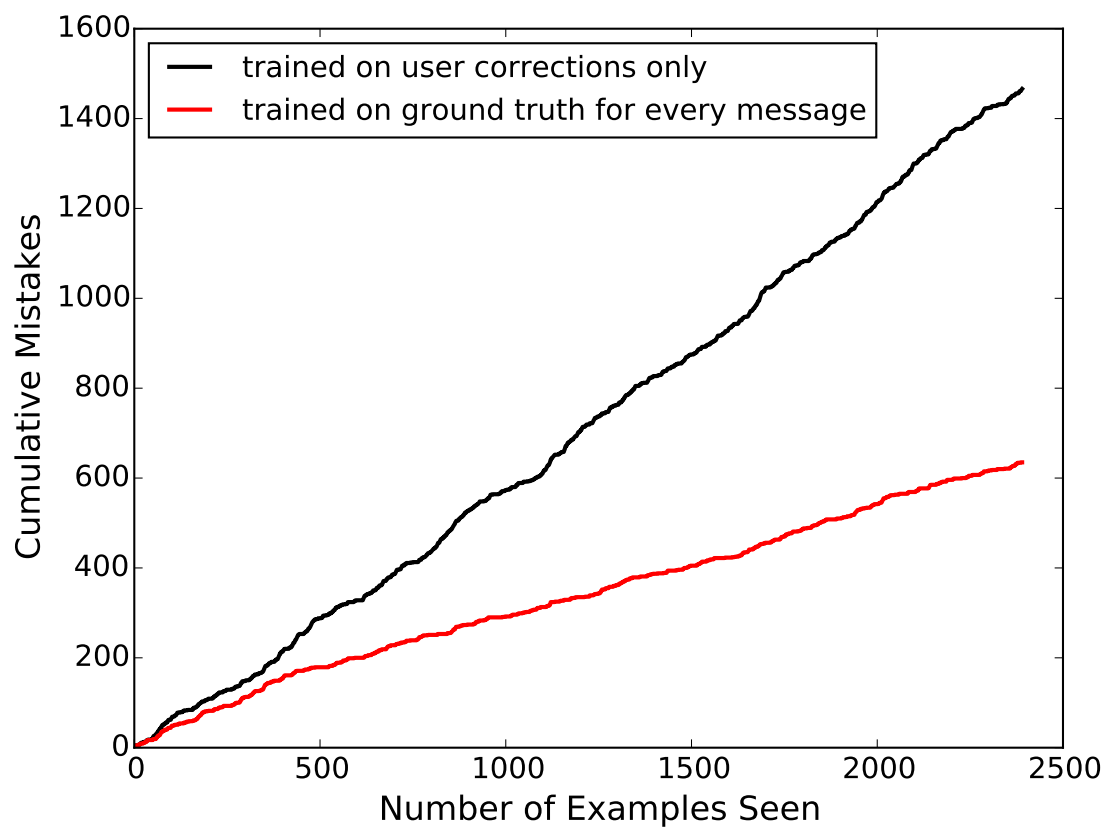


Figure 2.1: TAPE online tagging performance when the system is trained on user corrections only and when trained on ground truth tags for every message.



e.g., splitting a tag into multiple tags. The feature distribution for a particular tag can also change over time. Therefore, it is important for the learning agent to keep training, with or without user corrections. Lack of training can cause an online email classifier to lose its prompt adaptation capability, resulting in a performance decrease. This loss of performance is demonstrated in figure 2.1 that shows TAPE performance on a data set of 4982 email messages (described in Section 5.1.1). The first model follows the standard automatic email-tagging use case (*trained-on-correction*): each message is first predicted by the email predictor, and then the user corrects the tags, if necessary.<sup>2</sup> When the user corrects a tag, only then the underlying model is updated. In contrast, the second model follows the *standard online* learning protocol: each message is first predicted by the email predictor, and then its correct tags are given to the predictor to update the classifier(s). In a real system, this would require the user to *always* confirm the correct tags for a message after they have been predicted. This standard online protocol, while probably not a plausible use case, sets an upper-baseline (i.e., best case performance) for the email predictor.

As shown in figure 2.1, for about the first 100 messages, the number of mistakes made by the trained-on-correction system closely follows the number of mistakes made by the standard online method. This is because both systems receive similar amounts of user-feedback when they have just begun to learn. The standard-online system continues to receive training on every message, but the amount of training for trained-on-correction system drops as the system starts making correct predictions. This lack of training causes the trained-on-correction system to make more mistakes than the full-training standard-online system.

The gap between the mistakes plot in Figure 2.1 suggests that the trained-on-correction system can be improved by additional training. Therefore, we need a system that confirms tags on a message without explicitly asking the user. This can be achieved by making inferences drawn from unobtrusive observations of user behavior, formally called implicit feedback. One good indicator of user attention to the predicted tags is when the user corrects a tag. For example, if the user adds or removes a particular tag  $t$  on a message, then the classifier for  $t$  gets direct positive or negative feedback respectively. This is also an indication that the user approved the remaining predicted tags on the message as relevant tags (all of the predicted tags are usually displayed right next to each other

---

<sup>2</sup>notice that the user may forget to add a missing tag or to remove an irrelevant tag.

on the email user interface, so when the user corrects one tag, they are highly likely to see the other tag(s) and all others as irrelevant tags for this message. This motivates us to explore *implicit feedback*, automatically inferred by monitoring user interactions with the email client, that can indicate the level of user attention to the tags, and hence, can provide instances for additional training.

Implicit feedback-based methods have been developed and exploited in information retrieval and recommendation systems. Those systems seek to determine whether the user found an item to be interesting versus uninteresting. The underlying model is that the user will spend more time reading documents that are more interesting among all the documents he/she is presented with. *InfoScope* [37] exploits this idea to learn user preferences for *Usenet* discussion group messages by combining explicit user-feedback with a few sources of implicit feedback such as whether the user read the message or not, whether the user saved the message or deleted it, whether the user followed-up on the message or not. *Morita et al.* conducted a user study over a six-week period with eight participants and showed that, just by monitoring *reading time*, it is possible to predict the user's degree of interest in particular Usenet articles [28]. A series of user studies, conducted in different domains, confirmed that user actions such as printing, forwarding, and scrolling a page have a strong positive correlation with the user's stated level of interest in the content of the presented article [23, 6, 15].

*Konstan et al.* found that actions such as printing, forwarding, and replying privately to a message are also good indicators of user interest [23]. *Claypool et al.* developed a custom browser that monitors time spent on a page, scrolling time, scrolling method, mouse movements, and keyboard activities, and observed a strong positive correlation between these implicitly captured activities with the user's stated level of interest [6]. *Fox et al.* conducted a user study in which 146 participants used a custom browser, capable of recording a set of 30 implicit measures of user activity, in their regular work for six days [15]. The results show a direct relationship between the implicit measures and the user's explicit satisfaction ratings.

Implicit feedback can also indicate the user's relative preferences over information. In this case, the underlying model is that if the user clicks on an item in a ranked list of search results (assuming that the user scans the list sequentially), the items above it in the list were less interesting than the selected item. *Radlinski et al.* [31] extended the Osmot search engine [32] based on this idea. The extended engine combines eye tracking

and user-click data to update the ranking function, and this method is shown to be very robust to noise in user behavior.

Implicit feedback methods have also been reported to be very effective in learning user preferences for online dating [30, 2], online music recommendation services [20, 29], interactive video retrieval [38], a context-aware recommender service for mobile applications [11], and a job recommender service [19]. Although it is commonly believed that implicit feedback can only indicate positive preferences, *Lee et al.* [26] developed a job recommender service that improves the quality of recommendation by using implicit negative feedback (e.g., the user opened a job description but did not save). All of these papers identified *Cold Start* as a common problem. A new user in a system finds it boring to provide many explicit ratings but the learning algorithm requires many ratings to start predicting [42]. Implicit feedback in such cases has been proven to be helpful from the very beginning, and which reduces user frustration.

Although most of the work on implicit feedback research has been devoted to improving web-based applications, a recent realization among the research community is that the same approaches can contribute to developing smart desktop systems [5]. *Chirita et al.* propose a system that clusters desktop documents based on access timestamps, the number of steps between consecutive accesses of different files, and the time window they are accessed within [5]. The goal is to exploit the clusters to define desktop usage contexts and suggest context-related documents to the user.

## Chapter 3: TAPE Implicit Feedback System

### 3.1 TAPE Email Predictor

The TAPE Email Predictor is a combination of a Microsoft Outlook add-in and a Java backend server. The Outlook add-in provides the user interface for performing tag operations, and it passes the data to the server to perform learning and prediction. Users define their own tag sets. The number of tags for long-term users of the system ranges from 50 to 350. To perform multi-label prediction, TAPE learns one CW linear classifier per tag. An email message with 3 user-assigned tags creates positive examples for 3 of these classifiers and negative examples for all of the others. We chose the CW classifier because of its aggressive update behavior. When the user corrects a classifier mistake on an email message, the classifier makes an “aggressive” update so that the email message would have been correctly classified with a large margin. This makes the classifier very responsive to user feedback, which is important to the user experience. A drawback is that the classifier is very sensitive to mislabeled training data. If the user flubs, for example by deleting the wrong tag or mistyping a tag, then the classifier will make a big change in its parameters. To address this, TAPE contains an automatic undo facility, so that if the user immediately corrects the flub, TAPE detects this and unwinds the parameter change [36].

When an email message arrives, it is processed to extract a binary feature vector. Features include information about the sender, the set of recipients, the words in the subject line, and the words in the email body (after removing HTML markup and stopwords and performing stemming). The total number of features grows over time as new words and new email addresses are encountered. A typical feature vector contains 30,000-50,000 potential features of which only 30-200 are “turned on”. The feature vector is then passed to the tag classifiers, each of which produces a predicted margin. A common practice in machine learning is to make a positive prediction when the margin is greater than zero and a negative prediction otherwise. However, this zero-threshold based prediction does not say anything about the confidence in prediction, and also does not help rank the

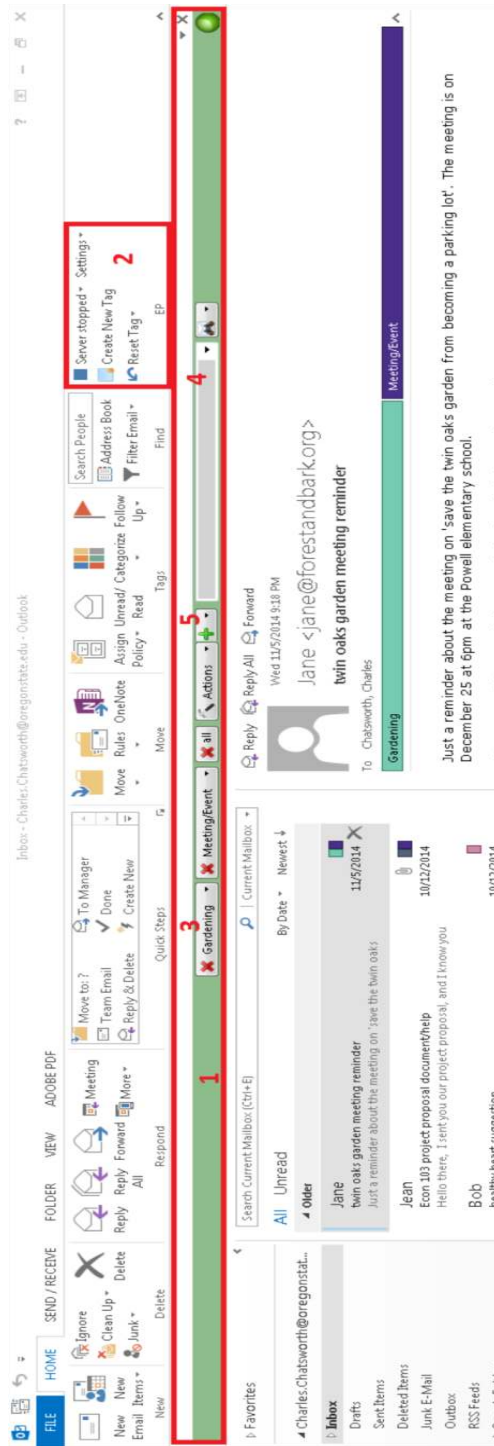


Figure 3.1: TAPE Email Predictor Tag Interface on Microsoft Outlook.

tags with respect to their relevance with the message. Therefore, we map the margins onto the  $[0,1]$  probability scale by applying a sigmoid (logistic) transformation. All tags with predicted probability above a threshold (default to 0.7) are added to the email message and displayed to the user. If there are no such confidently-predicted tags, then the tag with the highest probability is “promoted” and added to the message. However, if all tags have predicted probability below 0.01 (we call it the noise floor), then no tags are promoted. Implicit feedback will be most useful if it can provide confirmation that the promoted tags are indeed correct. The PREDICT function defined in Algorithm 1 summarizes the described steps in predicting tags for a message.

Figure 3.1 shows the main components of the TAPE user interface. The tags assigned to a message are shown on the TaskBar (#1 in Figure 3.1) as click buttons (#3 in Figure 3.1). The user can easily remove a tag by clicking on the left side (on the red cross) of the button. To add a tag, the user has several options. First, the drop-down “combo” search box (#4 in Figure 3.1) allows the user to type the name of a tag. As the user types, matching tags appear in a drop-down menu, and the user can use the mouse or arrow keys to select the desired tag. For users with few tags, it is typically more convenient to click on the drop-down arrow and select the desired tag from the menu, as the menu is large enough to show at least 15 tags. A second approach is to use the drop-down menu near the “plus” sign (#5 in Figure 3.1). This dropdown provides two ways to add a tag. The lower part of the dropdown provides a menu of the 12 most-recently used (MRU) tags. The upper part of the dropdown provides a menu of the top five tags whose confidence was below the 0.7 prediction threshold. The user can add tags by clicking on entries from either menu. These features are most helpful for users with many tags. The user can also create a new tag by clicking on the “+” button and typing the name of a tag.

The UI contains a few other components. A small control box (#2 in Figure 3.1) allows developers to stop and start the TAPE backend server. The user can also request updated predictions by selecting one or more email messages, right-clicking, and choosing “Predict tags for this message”.

## 3.2 User Interface Instrumentation

To support implicit feedback, we added instrumentation to TAPE to capture and record information about the user’s interaction with email messages in Microsoft Outlook. For each message, we computed the total number of times each of the following events occurred:

- message was opened and read in either the Outlook Explorer or the Outlook Inspector
- user added or removed a tag on the message
- user added or removed a flag from the message
- user moved the message to a folder
- user copied, replied, forwarded, or printed a message
- user saved an attachment from the message

We will refer to these events collectively as “implicit feedback events”. Some of them require additional explanation as follows. The Outlook Explorer corresponds to the user interface shown in Figure 3.1, which displays a short summary of each email message and provides a “viewing pane” that displays the currently selected message. The Outlook Inspector displays a single email message in a separate window. Some users prefer the Explorer, others prefer the Inspector. Outlook allows the user to set various flags on a message such as “follow up today”, “follow up tomorrow”, “completed”, and so on. Our instrumentation also tracks the total reading time for each message (i.e., the total time the window containing the message is in focus).

## 3.3 Baseline Algorithms

We designed four implicit feedback algorithms and three baseline algorithms for comparison. In this section, we describe the baseline algorithms.

### 3.3.1 No Implicit Feedback

The No Implicit Feedback (NoIF) algorithm never creates implicit training examples. It only creates a training example for a tag if the user adds or removes that tag from the email message. This is the standard behavior of TAPE, and it is a baseline against which to compare the other algorithms.

The prediction and the training procedures of NoIF are shown in Algorithm 1. The PREDICT function returns the predicted tags for a message. The prediction can be either the above threshold tags, the promoted tag, or no tag at all in the case when all the predicted probabilities are below the noise floor. The TRAINONUSERCORRECTION function shows how the classifiers are updated when the user either adds or removes a tag. When the user adds a tag to a message, the message is treated as a positive training example for that tag and the corresponding classifier is trained accordingly. In the case there is no classifier already defined for the tag (e.g., user creates a new tag and adds it to a message), a new classifier is instantiated and then trained positively on the message. Similarly, when the user removes a tag from a message, the message is treated as a negative training example for that tag and the corresponding classifier is updated. In either case, only the tags that the user corrects on a message are trained on that message. The amount of training in NoIF is therefore equal to the number of corrections made by the user.

### 3.3.2 Self Training

In Semi-Supervised Learning (SSL), self training is one of the first algorithms that exploits unlabeled data [1, 34]. The wrapper algorithm, with an underlying supervised learning method, is an iterative algorithm that starts by training only on the labeled data. Then in each iteration, it predicts class labels for the unlabeled data and then trains on its own high-confidence predictions. The underlying assumption is that the high confidence predictions of the learning algorithm are likely to be correct, and therefore, those predictions can safely be added to the labeled data for the next iteration of training.

We adapt the self training (STrain) method to our email tagging problem by introducing two parameters: Positive Prediction Training Lower Threshold ( $PL$ ) and Negative



---

**Algorithm 1** NoIF
 

---

```

1: procedure PREDICT( $e, \theta, nf$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor
3: Output:  $predicted\_tags = \{(t_i, p_i) | \text{tag } t_i \text{ predicted for the message } e \text{ with probability } p_i\}$ 
4:    $predicted\_tags = \{\}$ 
5:   for each tag  $t_i$  do
6:     compute the probability  $p_i$  that tag  $t_i$  should be assigned to  $e$ 
7:     if  $p_i \geq \theta$  then
8:        $predicted\_tags \leftarrow predicted\_tags + (t_i, p_i)$ 
9:   if  $predicted\_tags = \{\}$  then
10:     $t = \arg \max_{t_i} p_i$ 
11:     $p = \max_i p_i$ 
12:    if  $p \geq nf$  then
13:       $predicted\_tags \leftarrow (t, p)$ 
14:   Return  $predicted\_tags$ 
15:
16: procedure TRAINONUSERCORRECTION( $e, tcorr, t$ )
17: Input:  $e$  = email message,  $tcorr$  = user tag correction activity (add/remove) on
    message  $e$ ,  $t$  = tag that was corrected
18: Output: updated classifier  $c$  for tag  $t$ 
19:   if  $tcorr = \text{add tag}$  then
20:     if no classifier for tag  $t$  exists then
21:       instantiate a classifier for tag  $t$ 
22:     Let  $c$  be the classifier for tag  $t$ 
23:     TrainPositive( $c, e$ )
24:   else
25:     TrainNegative( $c, e$ )
26:   Return  $c$ 

```

---

---

**Algorithm 2** STrain
 

---

```

1: procedure PREDICT( $e, \theta, nf, pl, pu = 1.0, nl = 0.0, nu$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor,
3:  $pl$  = positive prediction training lower threshold
4:  $pu$  = positive prediction training upper threshold
5:  $nl$  = negative prediction training lower threshold
6:  $nu$  = negative prediction training upper threshold
7: Output:  $predicted\_tags$  = a set of tags predicted for the message  $e$ 
8:    $predicted\_tags \leftarrow$  NOIF_PREDICT( $e, \theta, nf$ )
9:   for each  $(t, p)$  in  $predicted\_tags$  do
10:     Let  $c$  be the classifier for tag  $t$ 
11:     if  $pl \leq p \leq pu$  then TrainPositive( $c, e$ )
12:     if  $nl \leq p \leq nu$  then TrainNegative( $c, e$ )
13:   Return  $predicted\_tags$ 
14:
15: procedure TRAINONUSERCORRECTION( $e, tcorr, t$ )
16: Input:  $e$  = email message,  $tcorr$  = user tag correction activity (add/remove) on
    message  $e$ ,  $t$  = tag that was corrected
17: Output: updated classifier  $c$  for tag  $t$ 
18:    $c \leftarrow$  NOIF_TRAINONUSERCORRECTION( $c, tcorr, t$ )
19:   Return  $c$ 

```

---

Prediction Training Upper Threshold ( $NU$ ). These two thresholds control how confident a positive or a negative tag prediction has to be for the algorithm to train on it. After making a prediction, STrain creates a positive training example for the tags that are predicted with probability above the specified threshold  $PL$  and a negative training example for the tags that are predicted with probability below the specified threshold  $NU$ . The thresholds  $PL$  and  $NU$  are learned using a validation data set. As shown in Algorithm 2, if the user corrects a tag, the classifiers are trained exactly as in the NoIF algorithm.

### 3.3.3 Online Learning

This algorithm mimics the online prediction framework studied in theoretical analysis of machine learning algorithms. The algorithm ignores all implicit feedback events. Instead, immediately after the tags are predicted for a message, the algorithm replaces the predicted tags with the correct tags and creates training examples for them. Hence, it provides perfect feedback to the TAPE multi-label classifier. In a real system, this would require the user to *always* confirm the correct tags for a message after they have been predicted. This online algorithm (shown in algorithm 3), while probably not a plausible use case, sets an upper-baseline for the TAPE email predictor.

---

#### Algorithm 3 Online

---

```

1: procedure PREDICT-AND-ONLINE-TRAIN( $e, \theta, nf, G$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor,  $G$ 
   = the set of ground truth tags for the message  $e$ 
3: Output:  $predicted\_tags = \{(t_i, p_i) | \text{tag } t_i \text{ predicted for the message } e \text{ with probability } p_i\}$ 
4:    $predicted\_tags \leftarrow \text{NOIF\_PREDICT}(e, \theta, nf)$ 
5:   Let  $T$  be the set of all tags and  $C$  be the set of the corresponding classifiers
6:   for each tag  $t \in T$  do
7:     Let  $c \in C$  be the classifier for tag  $t$ 
8:     if  $t \in G$  then
9:       TrainPositive( $c, e$ )
10:    else
11:      TrainNegative( $c, e$ )
12:   Return  $predicted\_tags$ 

```

---

### 3.4 Implicit Feedback Algorithms

In this section, we describe the proposed implicit feedback algorithms. Each of these algorithms is designed to be invoked every time a new implicit feedback event occurs. The algorithm then decides whether enough implicit feedback has been received to conclude that the current set of tags on the message is correct. If so, it creates positive training examples for each tag on the message and negative training examples for each tag that is not on the message. This training is invisible to the user, and in fact, if the user subsequently changes a tag, the automatic undo facility will roll back the implicit training and train on the explicit feedback provided by the user.

#### 3.4.1 Simple Implicit Feedback

When the user changes any tag (by deleting or adding a tag), the Simple Implicit Feedback (SIF) algorithm immediately treats all remaining tags as correct, and creates implicit feedback training examples (positive examples for the tags that are present and negative examples for all tags that are absent). The rationale for this is that if the user makes any changes to the tags, then the user has attended to the tags. Hence, any tags that the user does not change are highly likely to be correct. This increases the total amount of training the TAPE Email Predictor receives, especially when the label cardinality of the data set is high. If the label cardinality (defined as the average number of tags per message) is closer to 1, then messages are likely to have only one predicted tag. Therefore, upon user-correction, SIF will generate many negative training examples but not many positive training examples. Algorithm 4 summarizes the SIF training procedure.

Note that SIF is aggressive. If the user corrects one tag, SIF immediately trains all the classifiers. If the user makes any subsequent tag changes, these are correctly handled by the undo system, as described above.

#### 3.4.2 Implicit Feedback without SIF

Implicit Feedback without SIF (IFwoSIF) algorithm maintains a count of the total number of implicit feedback events, computed in the `COMPUTEIFSCORE` function shown in Algorithm 5. It treats tag changes just like all other implicit feedback events. When this count exceeds a specified threshold, then it creates the implicit feedback training

---

**Algorithm 4** SIF
 

---

```

1: procedure PREDICT( $e, \theta, nf$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor,
3: Output:  $predicted\_tags$  = a set of tags predicted for the message  $e$ 
4:    $predicted\_tags \leftarrow$  NOIF_PREDICT( $e, \theta, nf$ )
5:   Return  $predicted\_tags$ 
6:
7: procedure TRAINONUSERCORRECTION( $e, tcorr, t, mT$ )
8: Input:  $e$  = email message,
9:  $tcorr$  = user tag correction activity (add/remove) on message  $e$ ,
10:  $t$  = tag that was corrected,  $mT$  = the set of tags currently on the message
11: Output: the set of classifiers  $C$  updated after the user correction
12:   if  $tcorr$  = add tag then
13:     if no classifier for tag  $t$  exists then
14:       instantiate a classifier for tag  $t$ 
15:       Let  $c$  be the classifier for tag  $t$ 
16:       TrainPositive( $c, e$ )
17:     else
18:       TrainNegative( $c, t$ )
19:
20:   Let  $T$  be the set of all tags and  $C$  be the set of the corresponding classifiers
21:   for each tag  $\hat{t} \in T$  do
22:     Let  $\hat{c} \in C$  be the classifier for tag  $\hat{t}$ 
23:     if  $\hat{t} \neq t$  and  $\hat{t} \in mT$  then
24:       TrainPositive( $\hat{c}, e$ )
25:     if  $\hat{t} \neq t$  and  $\hat{t} \in T \setminus mT$  then
26:       TrainNegative( $\hat{c}, e$ )
27:   Return  $C$ 

```

---

---

**Algorithm 5** IFwoSIF
 

---

```

1: procedure PREDICT( $e, \theta, nf$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor
3: Output:  $predicted\_tags = \{(t_i, p_i) | \text{tag } t_i \text{ predicted for the message } e \text{ with probability } p_i\}$ 
4:    $predicted\_tags \leftarrow \text{NOIF\_PREDICT}(e, \theta, nf)$ 
5:   Return  $predicted\_tags$ 
6:
7: procedure TRAINONUSERCORRECTION( $e, tcorr, t$ )
8: Input:  $e$  = email message,  $tcorr$  = user tag correction activity (add/remove) on
   message  $e$ ,  $t$  = tag that was corrected
9: Output: updated classifier  $c$  for tag  $t$ 
10:   $c \leftarrow \text{NOIF\_TRAINONUSERCORRECTION}(e, tcorr, t)$ 
11:  Return  $c$ 
12:
13: procedure TRAINONUSERACTIVITY( $e, ifactivity, \theta, mT$ )
14: Input:  $e$  = email message,  $ifactivity$  = IF activity on message  $e$ ,  $\theta$  = IF threshold,
15:  $mT$  = the set of tags currently on the message
16: Output: the set of classifiers  $C$  updated after the user correction
17:   $ifscore = \text{COMPUTEIFSCORE}(ifactivity)$ 
18:  Let  $T$  be the set of all tags and  $C$  be the set of the corresponding classifiers
19:  if  $ifscore \geq \theta$  then
20:    for each tag  $t \in T$  do
21:      Let  $c \in C$  be the classifier for tag  $t$ 
22:      if  $t \in mT$  then
23:        TrainPositive( $c, e$ )
24:      else
25:        TrainNegative( $c, e$ )

```

---

examples. Recall that we also keep track of the total reading time of each message. An examination of the reading time data showed that it had some very large values, which presumably occur when the user opens an email message but then engages in other activities that do not involve the computer. Consequently, with the exception of the IFwSIFLW algorithm that we will describe below, we did not use reading time in our implicit feedback algorithms.

The function TRAINONUSERACTIVITY in Algorithm 5 shows the steps to create the positive training examples for the tags that are on the message and the negative training examples that are not on the message. The threshold is learned using a validation data set. If the user corrects a tag on the message, then the function TRAINONUSERCORRECTION proceeds exactly as in the NoIF algorithm. The PREDICT function is also the same as in the NoIF algorithm.

---

**Algorithm 6** IFwSIF
 

---

```

1: procedure PREDICT( $e, \theta, nf$ )
2: Input:  $e$  = incoming email message,  $\theta$  = prediction threshold,  $nf$  = noise floor
3: Output:  $predicted\_tags = \{(t_i, p_i) | \text{tag } t_i \text{ predicted for the message } e \text{ with probability } p_i\}$ 
4:    $predicted\_tags \leftarrow \text{NOIF\_PREDICT}(e, \theta, nf)$ 
5:   Return  $predicted\_tags$ 
6: _____
7: procedure TRAINONUSERCORRECTION( $e, tcorr, t, mT$ )
8: Input:  $e$  = email message,
9:  $tcorr$  = user tag correction activity (add/remove) on message  $e$ ,
10:  $t$  = tag that was corrected,  $mT$  = the set of tags currently on the message
11: Output: the set of classifiers  $C$  updated after the user correction
12:    $C \leftarrow \text{SIF\_TRAINONUSERCORRECTION}(e, tcorr, t, mT)$ 
13:   Return  $C$ 
14: _____
15: procedure TRAINONUSERACTIVITY( $e, ifactivity, \theta, mT$ )
16: Input:  $e$  = email message,  $ifactivity$  = IF activity on message  $e$ ,  $\theta$  = IF threshold,
17:  $mT$  = the set of tags currently on the message
18: Output: the set of classifiers  $C$  updated after the user correction
19:    $C \leftarrow \text{IFWOSIF\_TRAINONUSERACTIVITY}(e, ifactivity, \theta, mT)$ 
20:   Return  $C$ 

```

---

### 3.4.3 Implicit Feedback with SIF

This algorithm combines SIF and IFwoSIF. If the user changes a tag, then implicit feedback examples are immediately created. Otherwise, IFwoSIF continues to count up events until the number of implicit feedback events exceeds the learned threshold, at which point it creates the implicit feedback examples. This algorithm also does not use total reading time for the same reason described in case of IFwoSIF. Algorithm 6 shows the PREDICT, TRAINONUSERCORRECTION and TRAINONUSERACTIVITY functions for IFwSIF. The threshold is learned using a validation data set.

### 3.4.4 Implicit Feedback with SIF using Learned Weights

So far we have assumed that the implicit feedback counts can simply be added together, and a threshold can be defined to compare against the total sum. However, the relative importance of the different types of events may not be equal. For example, correcting a tag on a message may be a better indicator of user attention to tags than replying to a message.

An alternative to treating all IF events as equally important is to learn a weight for each type of event and then compare the weighted sum of events against a threshold. This leads us to a binary classification problem where the training example  $i$  has the form  $(\langle x_{i1}, \dots, x_{iT_i} \rangle, y_i)$ .  $x_{it}$  is a feature vector that summarizes all of the events observed up to time  $t$ . The subscript  $T_i$  is the time of the last observed implicit feedback event for instance  $i$ . When collecting training data, we assume we have a source of the true (‘gold’) labels, so that  $y_i = +1$  if the predicted tag was correct and  $y_i = -1$  if it was incorrect. Let  $P = \{i | y_i = +1\}$  and  $N = \{i | y_i = -1\}$  denote the positive and negative training examples, respectively. Our goal is to learn a function such that if  $y_i = 1$ , then there exists a time  $t^*$  such that  $\forall_{t \geq t^*} f(x_{it}) > 0$  and if  $y_i = -1$  then for all time  $t$ ,  $f(x_{it}) < 0$ .

Let us assume that the features are all measures of user attention that are non-negative and that grow over time. Suppose we parameterize  $f$  as a linear function  $f(x) = \vec{w} \cdot x + b$  where each element  $w_j$  of  $w$  is non-negative:  $\forall_j w_j \geq 0$ . Then  $f$  will be monotonically increasing. Conceptually,  $f(x_{it})$  grows until it exceeds the threshold  $-b$ , at which point, we conclude that the predicted tags are true positives and all non-predicted tags are true negatives. TAPE can then train on the message. If  $f(x_{iT_i}) < 0$  at the point where we



stop collecting implicit feedback on example  $i$ , then do not have enough implicit feedback information to trigger training.

Given the monotonic behavior of  $f$ , each training example can be converted to the following set of constraints:

$$\forall_{i \in P} w \cdot x_{iT_i} > 0 \quad (3.1)$$

$$\forall_{i \in N} w \cdot x_{iT_i} < 0. \quad (3.2)$$

In addition, we would like to encourage early detection of the positives, so we want to maximize  $\sum_t w \cdot x_{it}$  for the positive instances  $t \in P$ .

With this objective, we formulate the following optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C_N \sum_{i \in N} \xi_i + C_P \sum_{i \in P} \xi_i + C_B \sum_{t, i \in P} v_{it} \quad (3.3)$$

subject to

$$\forall_i y_i [\mathbf{w} \cdot x_{iT_i} + b] \geq 1 - \xi_i \quad (3.4)$$

$$\forall_{i \in P} \forall_t [\mathbf{w} \cdot x_{it} + b] \geq 1 - v_{it} \quad (3.5)$$

$$\forall_i \xi_i \geq 0 \quad (3.6)$$

$$\forall_{it} v_{it} \geq 0 \quad (3.7)$$

$$\forall_j w_j \geq 0 \quad (3.8)$$

$C_N$  indicates how important it is to fit all of the negatives. This should be large to ensure no false positives.  $C_P$  indicates how important it is to fit the positive constraints. This should also be large (perhaps =  $C_N$ ) to ensure that we eventually get each positive right. Finally,  $C_B$  is an added bonus for getting “earlier” positives right.

Given some training data, we can learn the weights on the implicit feedback events by solving this convex optimization problem. We can then use those weights in the TRAINONUSERACTIVITY function in Algorithms 5 and 6 to compute the weighted sum

of the event counts to obtain the implicit feedback score on each message. The Implicit Feedback with SIF using Learned Weights (IFwSIFLW) model is therefore exactly the same as IFwSIF (algorithm 6), except that IFwSIFLW exploits these learned weights to compute the implicit feedback score (in contrast, IFwSIF is effectively setting all weights to 1 and only learning the threshold).

Because IFwSIFLW is learning weights on the IF events, we can now include reading time and rely on the weight learning to determine the relative importance of reading time versus other discrete IF events. To deal with the noisy aspect of reading times, we do not use the total reading time as a feature. Instead, we create three features for reading time is less than 1 second, reading time is between 1 and 5 seconds, and reading time is above 5 seconds. The reading time for each message corresponds to setting one of these features to 1 and the others to zero. If a message is open for a long time (e.g., user opened the message but stepped away from the computer), this just corresponds to setting the  $>5s$  feature to 1.

Now that we have defined all of our models, we measure the performance of each algorithm in terms of the total number of tag prediction errors. The *Online* baseline method should give the best results, because it provides ideal feedback, while the *NoIF* method should give the worst results, because it provides no implicit feedback. The central question of this thesis is how well the implicit feedback algorithms are able to close the gap between *NoIF* and *Online*.

## Chapter 4: The Lab-controlled User Study

We conducted a user study to collect data on how users interact with automatically tagged email. Our goal in the study was to simultaneously encourage the study participants to correct the email tags while also engaging them in performing email-directed tasks so that they would frequently fail to notice incorrect predicted tags. Using the interaction data recorded from the participants, we then compared the implicit feedback algorithms by replaying the user interactions while manipulating the fraction of tags corrected by each participant.

### 4.1 Dataset of Tagged Email Messages

We created an email dataset from a variety of web sources. The messages in this dataset were chosen to reflect the email life of a knowledge worker – a student in this case. In our scenario, the student is enrolled in two courses, is actively involved in projects, regularly attends meetings and events, and also has some hobbies. The dataset contains 330 messages with some of the messages also having file attachments. Each message was tagged with one or more tags from a set of six possible tags based on the content of the message: Economics, Entertainment, Gardening, Health, Math and Meeting/Event. Another graduate student, Michael Slater and I performed the ground truth tagging. Conflicts were resolved by taking a majority vote. The average number of ground truth tags per message was 1.24. Table 4.1 shows the distribution of tags. The messages are very similar to what a typical student would receive in everyday life. Some of the messages are completely informational (e.g., a professor describing a homework assignment). Others contain a request asking the recipient to perform one of the following tasks: save the attachment(s) from the message, edit a saved file and attach it to a reply or a new outgoing email message, send (reply or forward) messages with or without attachments, find requested information on the web, copy it into an email message, and send it. Here is an example email message:

Sender: Bishop <bishop@appqualify.com>  
Subject: help! math midterm solution

hi,  
Can you please FORWARD me the midterm solution the professor sent a few days back?  
Somehow I don't find that email!

thanks  
-Bishop

Notice that the task is cued with capital letters so that the study participants can easily recognize it. The purpose of these tasks was to distract the study participants from focusing solely on the tagging task. In the instructions given to the participants, the primary emphasis was placed on performing these tasks, but email tagging was also requested and described in detail.

After ground-truth tagging, the messages were randomly divided into four sets as follows. (All sampling was stratified to ensure that the frequency of the tags was approximately balanced within each set.) First, the 330 messages were divided into a set (“Train60”) of 60 messages for training and a set (“Test270”) of 270 messages for testing. The Test270 set was further divided into three sets: “Task1” with 66 messages, “Task2” with 102 messages, and “Task3” with 102 messages. The mean number of tags per message was 1.27 for Train60 and 1.23 for Test270.

To prepare the messages for the user study, we trained TAPE on the ground truth tags for the messages in “Train60”. We then applied the learned multi-label classifier to predict tags on all of the “Test270” messages. After training on only 60 messages, the classifier is not very accurate. Consequently, the predicted tags contain many errors. This was intentional, because we wanted to give the participants many incorrect tags to correct.

## 4.2 The User Study

We conducted a lab-controlled user study with a total of 15 participants. Only adult email users who receive 20 or more email messages everyday and who regularly use tags, categories, labels, or folders to organize email, were recruited to participate in the study. We also collected information about the participants' current email usage and email orga-

Table 4.1: The distribution of tags in the email dataset. For each tag, this table shows the percentage of messages that were assigned that tag. This totals to more than 100% because a message may have multiple tags.

Tags	%messages
Economics	15
Entertainment	18
Gardening	19
Health	23
Math	17
Meeting/Event	31

nization methods. On average, the participants received 37 email messages per day. 87% of the participants regularly use Gmail or Google Apps as their primary email client, and the remaining 13% use Microsoft Outlook. 80% of the participants had employed at least some tags, categories, labels, or folders to organize their messages in the past two weekdays. Most of the participants (71%) regularly use labels or folders. They organize their email using some combination of manually-created rules, interactively assigning labels, and interactively moving messages to folders. About 50% of the participants regularly use tags or categories (the tags provided natively in Outlook). Here are a few comments from the participants about their email organization methods:

*“I transfer to a particular folder then work later.”*

*“I have created different folders in my mail like Research, personal, work etc. Depending on the type of email I receive, I label and move the particular mail to respective folder. For example if I receive any email regarding internship, I will move it to folder work, so that it will be easily accessible to me whenever required. I will do labeling, moving, tagging using the options which we get in gmail (labels, move to etc).”*

The study participants interacted with Microsoft Windows and Outlook via a remote virtual terminal connected to a Windows server. This allowed us to completely control the desktop environment during the study so that the participants were not interrupted by their regular email flow, chat windows, calendar notifications, and so on.

The study was conducted in three two-hour sessions on three separate days. The first session was divided in half. During the first half, the students were asked to use the TAPE UI to tag messages that had been preloaded into their inbox. These are the “Train60” messages, and they were presented without any tags. While adding appropriate tags on these messages, the users learned the intended meanings of the tags and the properties of the email messages. They also learned how to carry out the tasks requested by the messages (e.g., how to save attachments, add attachments, reply, forward, etc.). Most importantly, the participants learned about the student role they were playing in the study.

In the second half of the first session, the participants were presented with the “Task1” messages in the inbox along with the (error-prone) predicted tags. During this session, the participants were told to perform the tasks that are described in the email messages. In addition, they were also asked to correct any tags that they notice are incorrect.

In the second and third sessions, the participants were asked to work on the “Task2” and “Task3” messages, respectively.

At the end of each session, the participants filled out a Qualtrics questionnaire that required them to provide the tags they believed were correct for each email message. We will call these the “user ground truth” tags.

Out of 15 participants, one participant dropped out during the course of the study. Therefore, we only consider for subsequent analysis the data from the 14 participants who successfully completed all three sessions.

### 4.3 Post-study Simulation

When a participant adds or deletes a tag from an email message, we will say that the participant has provided “Explicit Feedback” (EF). An initial analysis of the data collected from the study showed that the participants did not provide very much explicit feedback. The mean percentage of messages for which they corrected tags was 16.3% (standard deviation 0.9%). This is much lower than we have observed for long-term users of TAPE, where the EF level varies between 60% and 90%. This shows that the participants focused primarily on performing the tasks specified by the email messages and paid less attention to our request that they fix incorrect tags. With such low levels of explicit feedback (and with many incorrect tags), interpreting implicit feedback becomes very difficult.

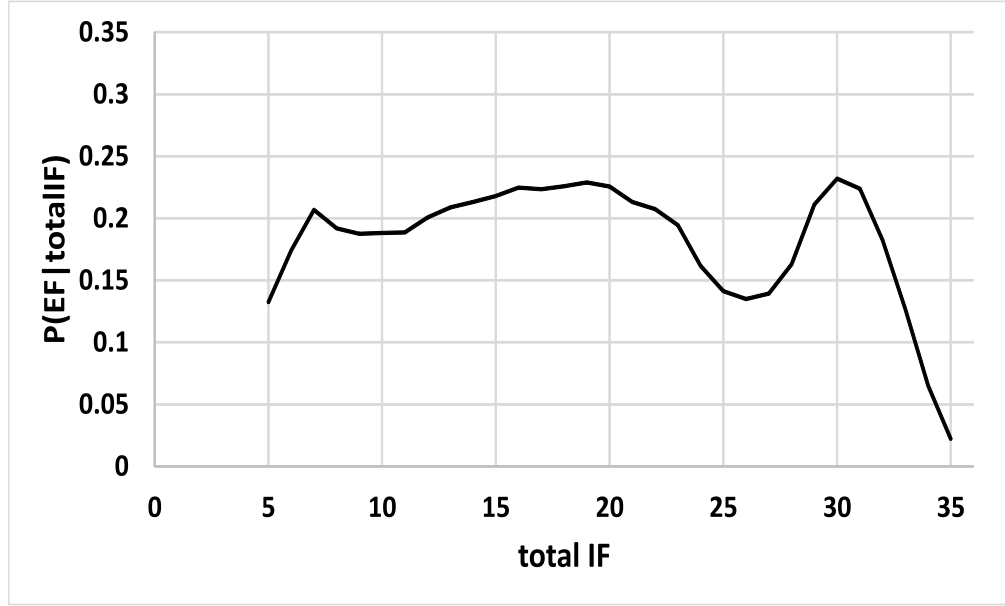


Figure 4.1: Conditional probability distribution,  $P(EF | totalIF)$ .

We addressed this shortcoming by combining the observed implicit feedback events with simulated explicit feedback as follows. For each participant and each email message, let the variable  $EF$  be 1 if the participant provided explicit feedback and 0 otherwise. Similarly, let the variable  $totalIF$  denote the total number of implicit feedback events observed for the participant on that email messages. From the set of observed  $(EF, totalIF)$  pairs, we can estimate the conditional probability  $P(EF = 1 | totalIF)$  that a randomly-selected participant will provide explicit feedback on a randomly-chosen email message given the number of implicit feedback events that they have produced. A smoothed version of this estimated distribution is shown in Figure 4.1. Note that the probability of providing EF is fairly constant (at around 0.2) as a function of the amount of IF, but drops off for very high levels of IF.

We then designed and implemented Algorithm 7 to modify the explicit feedback data  $\mathcal{D}^p$  collected for participant  $p$  to achieve a target level  $tEF$  of explicit feedback. The algorithm begins by constructing a vector of predicted explicit feedback probabilities  $\overline{pEF}$  based on the estimated distribution  $P(EF | totalIF)$  for participant  $p$ . Then it compares the fraction of observed EF to the target  $tEF$  and computes the number of

messages  $n$  whose EF must be changed. If the observed EF is too low, then explicit feedback is added to the  $n$  messages with highest predicted probability of EF. If the observed EF is too high, then explicit feedback is removed from the  $n$  messages with lowest predicted probability of EF.

---

**Algorithm 7** SampleEF( $p, tEF$ )

---

**Input:**  $p$  = User id for the participant,  
 $\mathcal{D}^p$  = User actions for messages for participant  $p$ ,  
 $P(EF|totalIF)$  = fitted probability of EF given total IF,  
 $tEF$  = target level of EF

**Output:**  $\mathcal{D}_{tEF}^p$  = User actions for participant  $p$  achieving  $tEF$

- 1:  $\overline{EF}$   $\leftarrow$  vector of observed EF for all messages
- 2:  $\overline{IF}$   $\leftarrow$  vector of observed IF for all messages
- 3:  $\overline{pEF}$   $\leftarrow$  vector containing  $P(\overline{EF}|\overline{totalIF})$  for messages
- 4:  $N$   $\leftarrow$  total # of messages for  $p$
- 5:  $EF$   $\leftarrow$  # observed messages with explicit feedback for  $p$
- 6:  $ObservedEF$   $\leftarrow$   $\frac{EF}{N}$  observed probability of EF for  $p$
- 7:  $n$   $\leftarrow$   $|ObservedEF - tEF| \cdot N$
- 8:     number of messages to change
- 9: **if**  $ObservedEF > tEF$  **then**
- 10:      $M$   $\leftarrow$   $n$  messages in increasing order of  $\overline{pEF}$
- 11:      $\mathcal{D}_{tEF}^p = \mathcal{D}^p$ , after removing EF from messages in  $M$
- 12: **if**  $ObservedEF < tEF$  **then**
- 13:      $M$   $\leftarrow$   $n$  messages in decreasing order of  $\overline{pEF}$
- 14:      $\mathcal{D}_{tEF}^p = \mathcal{D}^p$ , after adding EF to messages in  $M$
- 15: **Return**  $\mathcal{D}_{tEF}^p$

---

Algorithm 7 was applied to generate simulated event streams for each participant for  $tEF$  values of 0.2, 0.3, 0.4, 0.5, 0.7, and 0.8. These were then processed by TAPE as follows. First, TAPE was trained on the ground truth messages in “Train60” (exactly as the system was trained for the user study). Then the simulated events were processed by TAPE via a special “replay mode” using each of the seven implicit feedback algorithms. To assess performance, we measured the cumulative number and fraction of tags incorrectly predicted.

The IFwSIF and IFwoSIF algorithms employ a threshold to decide when to create training examples. To set that threshold, we randomly selected a few of the participants



and examined their simulated event data for different levels of target EF. We evaluated a few thresholds for each stream and took a majority vote to select the best threshold. We then use this best threshold (a total of 7 IF events) for all our experiments.

As described in Section 3.3.2, the STrain algorithm also has two threshold parameters: Positive Prediction Training Lower Threshold ( $PL$ ) and Negative Prediction Training Upper Threshold ( $NU$ ). Similar to the implicit feedback threshold, we set these  $PL$  and  $NU$  thresholds by randomly selecting a few of the participants and simulating their event data for different levels of target EF. We then evaluated different pair of values for these thresholds and took the majority vote to select the best pair of thresholds. We then use those values for  $PL$  ( $=0.74$ ) and  $NU$  ( $=0.08$ ) for all our experiments.

There was strong agreement on the range of good settings across participants and target EF values.

## 4.4 Results Analysis

Figure 4.3 summarizes the implicit feedback events collected from the study participants, summed over the three sessions. We employ a log scale in order to fit the wide range of values within a single plot. The narrow inter-quartile ranges of the box plots show that the distribution of these values is quite similar across the different participants. This reflects the fact that each participant was given the same set of email messages with the same tasks. The largest variation is observed in the number of tags changed and the number of times the email messages were opened (although the box for the latter appears small in the figure because of the log scale).

Figure 4.2 plots a timeline of the implicit feedback events for one study participant. We can see that the implicit feedback events are evenly distributed throughout the three study sessions.

As discussed above, without IF, the only machine learning choices are to train on all predictions or to train only on the EF. If we train on all predictions, the incorrectly-predicted tags will create bad training examples. If we train only on EF, we get fewer training examples, but they are all correct. An IF strategy seeks an intermediate path. It will succeed if there is a threshold on the number of IF events such that the loss in accuracy of the resulting incorrect training examples (created from “surviving” bad tags) is out-weighed by the gain from the resulting correct training examples. To evaluate this,

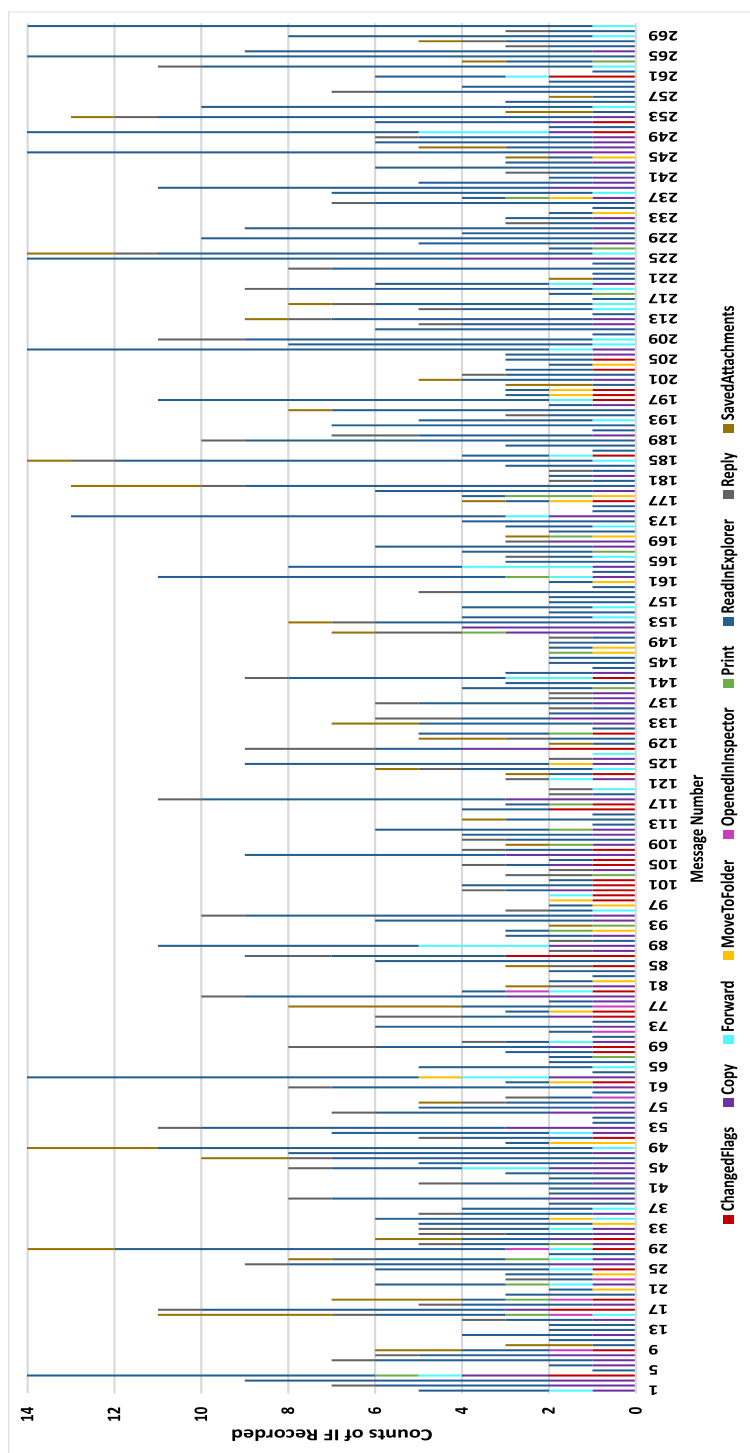


Figure 4.2: Implicit feedback captured during the study sessions of one participant. The first session ends after message 66, and the second session ends after message 168.

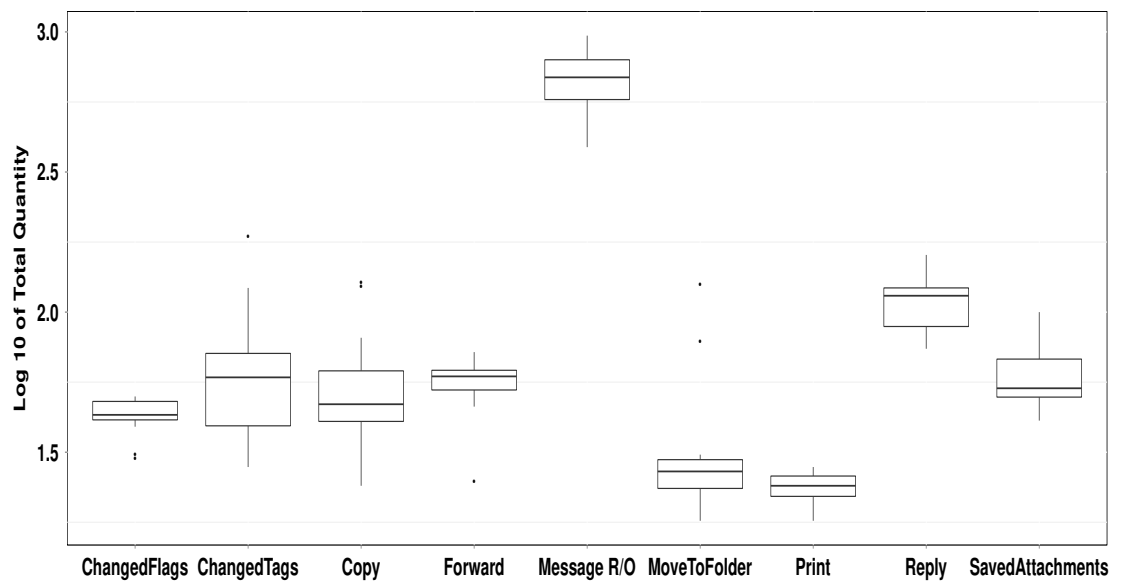


Figure 4.3: Total number of implicit feedback events captured (log scale) for each type of implicit feedback event. 'Message R/O' indicates the total number of times a message was opened in Outlook Explorer or in Outlook Inspector.

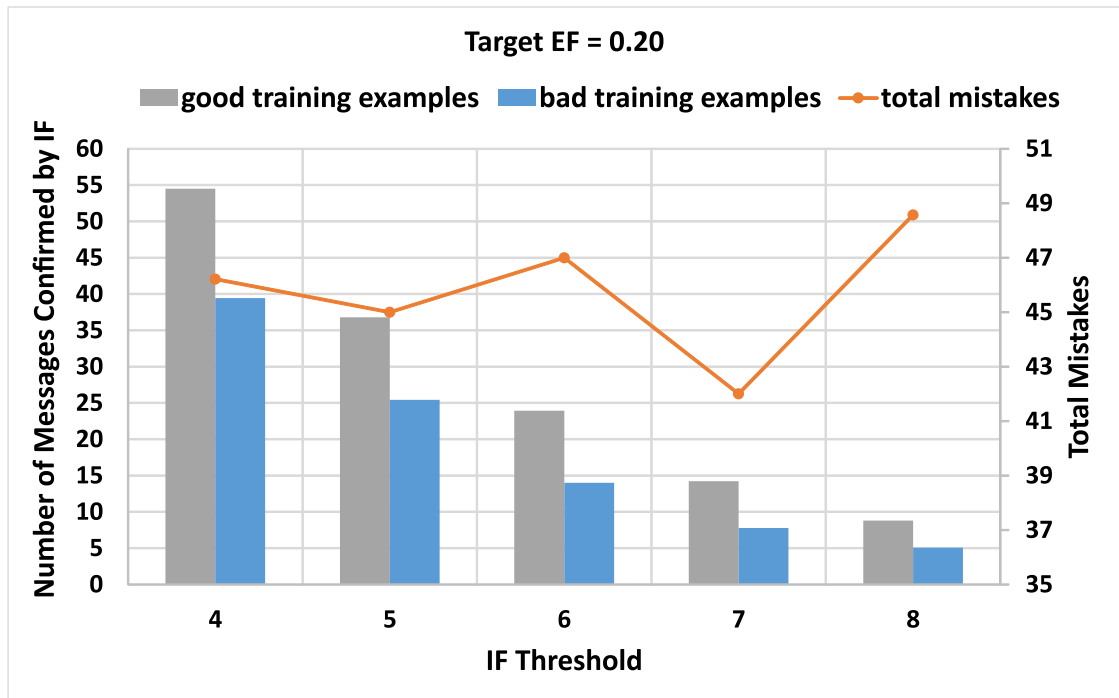
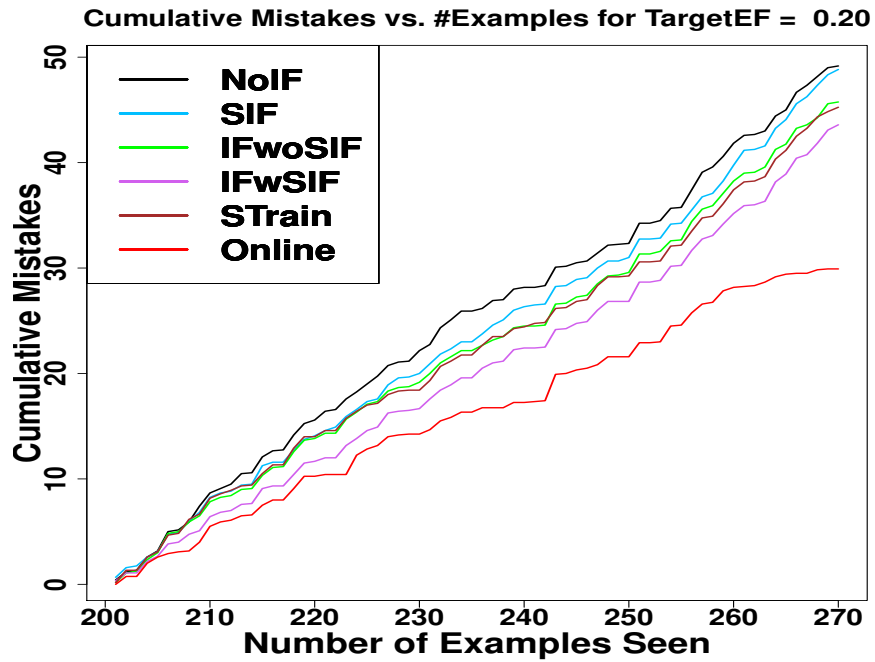
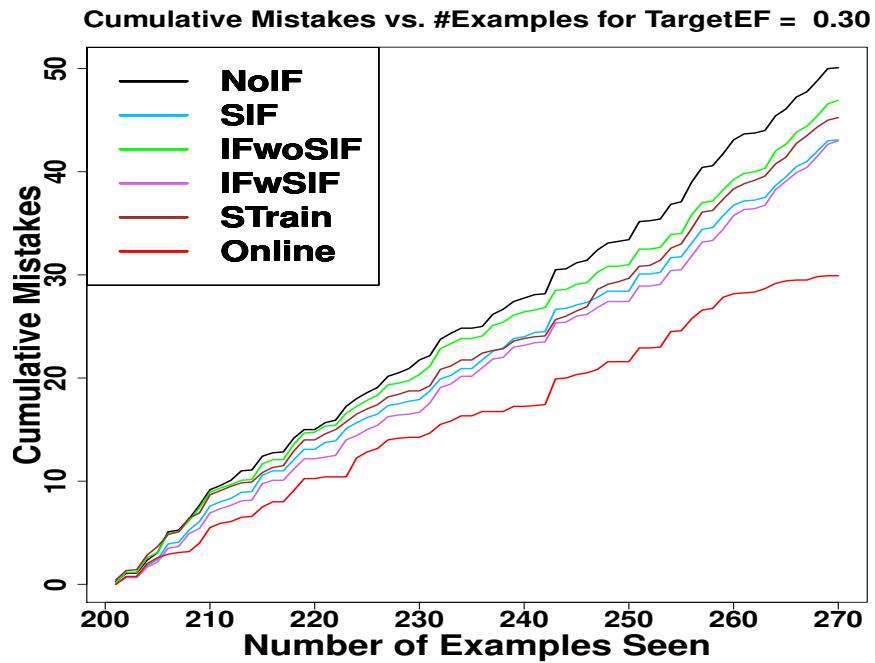


Figure 4.4: Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold (TargetEF = 0.20).

(a)  $t_{EF} = 0.20$ (b)  $t_{EF} = 0.30$

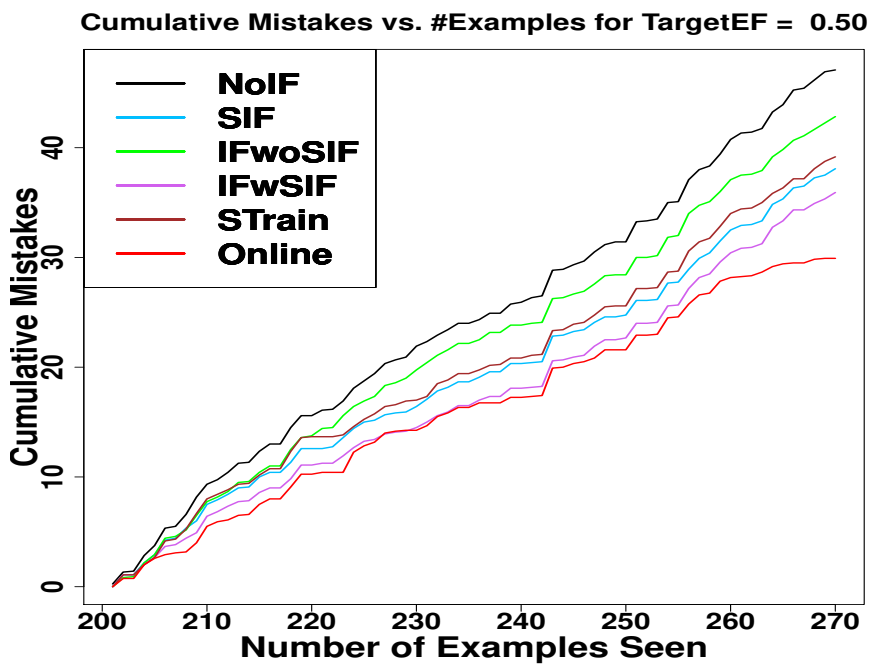
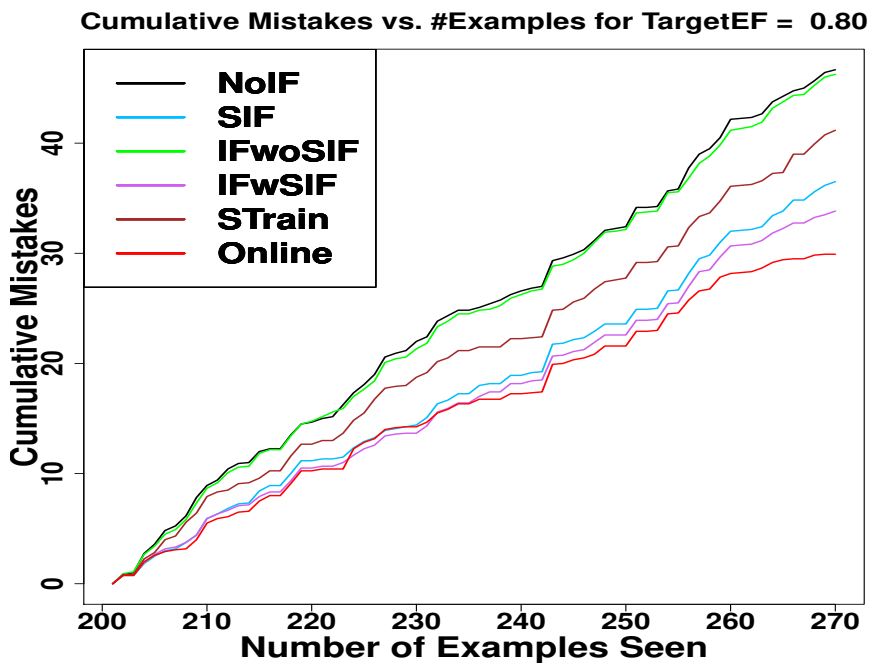
(c)  $tEF = 0.50$ (d)  $tEF = 0.80$ 

Figure 4.5: A comparison of the cumulative mistakes of each of the six IF algorithms on the last 70 email messages for six values of TargetEF.

we plot the number of bad and good training examples and the final cumulative number of mistakes as a function of the threshold (as shown in Figure 4.4 for target EF level of 0.20). Notice that the final cumulative number of mistakes in Figure 4.4 has a minimum (at IF threshold = 7.0).

Let us now consider the core question: Do the various implicit feedback algorithms improve classifier accuracy? We do not expect implicit feedback to provide much gain early in the experiment, because the predicted tags are not very accurate. As our goal is to assess the effectiveness of implicit feedback for long-term users of TAPE, we focus our analysis on the final 70 email messages. Figure 4.5 plots the cumulative prediction mistakes (averaged across all participants) of our algorithms for six levels of target EF. We exclude the IFwSIFLW model for the user study data because the user study data has only 270 messages, and separating some of these message for weight learning would make the test set very small.

A target EF of 0.20 is close to the actual behavior of the participants in the experiment. A target EF of 0.80 is typical of behavior exhibited by long-term users of TAPE, and a target EF of 0.50 is plotted to show an intermediate level of explicit feedback. In all cases, our baseline methods NoIF (no implicit feedback) and Online (complete online feedback) accumulate the largest and smallest number of errors, as expected. For target EF levels of 0.50 or more, IFwSIF produces the fewest errors, SIF is second best, followed by STrain, and IFwoSIF is the worst. This shows that simple implicit feedback (i.e., training as soon as the user changes any one tag) and implicit feedback (i.e., training when the number of implicit feedback events exceeds a threshold) both provide useful training examples. Combining them using IFwSIF gives better results than either method alone. For a target EF of 0.20, the implicit feedback algorithms do not produce very large error reductions compared to NoIF. But for a target EF of 0.80, the SIF and IFwSIF algorithms have largely eliminated the gap between NoIF and Online.

We had expected STrain to perform worse, but surprisingly, in many of the cases, STrain had very competitive performance. This may be because of the small size of the user study data set. Recall that the user study data set has a total of 330 messages and only 6 tags. We used 60 of the messages for pretraining, which means each tag classifier has seen 10 messages. Hence, the predictions in may have been mostly correct, and training on self-predictions ultimately helped the system. This may not be the case with a real data set with large tag-space and thousands of messages. We will explore this in

Table 4.2: Percentage tag prediction mistakes.

$\backslash$ target EF Alg.	0.20	0.30	0.40	0.50	0.70	0.80
NoIF	11.50±0.010	11.70±0.010	10.53±0.007	10.95±0.009	10.95±0.012	10.95±0.010
SIF	11.34±0.009	10.00±0.012	8.81±0.008	8.81±0.008	8.27±0.009	8.33±0.010
IFwoSIF	10.86±0.010	10.82±0.011	9.86±0.009	10.13±0.010	10.92±0.009	10.68±0.010
IFwSIF	<b>10.35±0.004</b>	<b>9.90±0.012</b>	<b>8.57±0.008</b>	<b>8.45±0.008</b>	<b>7.86±0.011</b>	<b>7.81±0.009</b>
STrain	10.57±0.008	10.20±0.018	8.80±0.013	9.10±0.008	9.15±0.009	9.70±0.009
Online	6.96±0.006	6.96±0.006	6.96±0.006	6.96±0.006	6.96±0.006	6.96±0.006

the next chapter when we analyze the case studies.

In all of the above plots, we have assumed perfect ground truth for the data. For example, if the user changes a tag, we assumed that the user corrects to the ground truth tag. However, in reality, users may make mistakes and may not be always consistent while tagging messages. In such cases, aggressively training on self-predictions will also be risky.

Table 4.2 provides another view of the mistakes in the above experiments. It reports the percentage of tag prediction mistakes on the last 70 email messages for each of the target levels of EF. Here we can see quantitatively that at a target EF of 0.20, the implicit feedback methods are giving only a small benefit over NoIF. But as the target EF level rises, 75% of the gap between NoIF and Online has been eliminated by IFwSIF.

TargetEF is the fraction of messages for which the simulated user examines and corrects tags. As TargetEF increases, the classifiers become more accurate and make fewer mistakes, which reduces the number of messages that require EF. Figure 4.6 plots the number of training examples (on the entire dataset) that the NoIF, SIF, and IFwSIF methods generate. Observe that SIF produces the predominant share of the training examples. Nonetheless, the additional examples added by implicit feedback have a substantial effect on further reducing prediction errors. This is indirect evidence that those examples are accurate. Using IFwSIF, the classifier receives 64% more training than NoIF and 14% more training than SIF (averaging across all levels of target EF).

Figure 4.7 provides additional insight into the quality of the implicitly-confirmed training examples. It reports the percentage of the confirmed email messages that were correctly confirmed by IFwSIF for various levels of target EF. We see that in all cases, implicit feedback is doing better than random. However, at a Target EF of 0.20, only



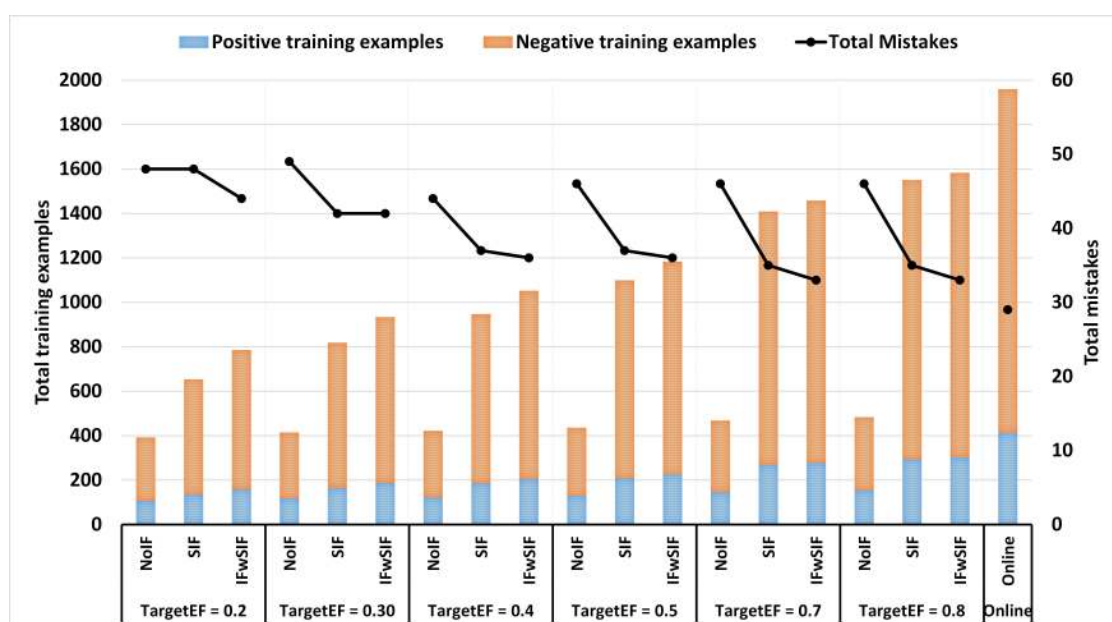


Figure 4.6: Total number of training examples for the entire experiment (left axis) and total number of prediction mistakes on the last 70 messages (right axis) for different levels of TargetEF.

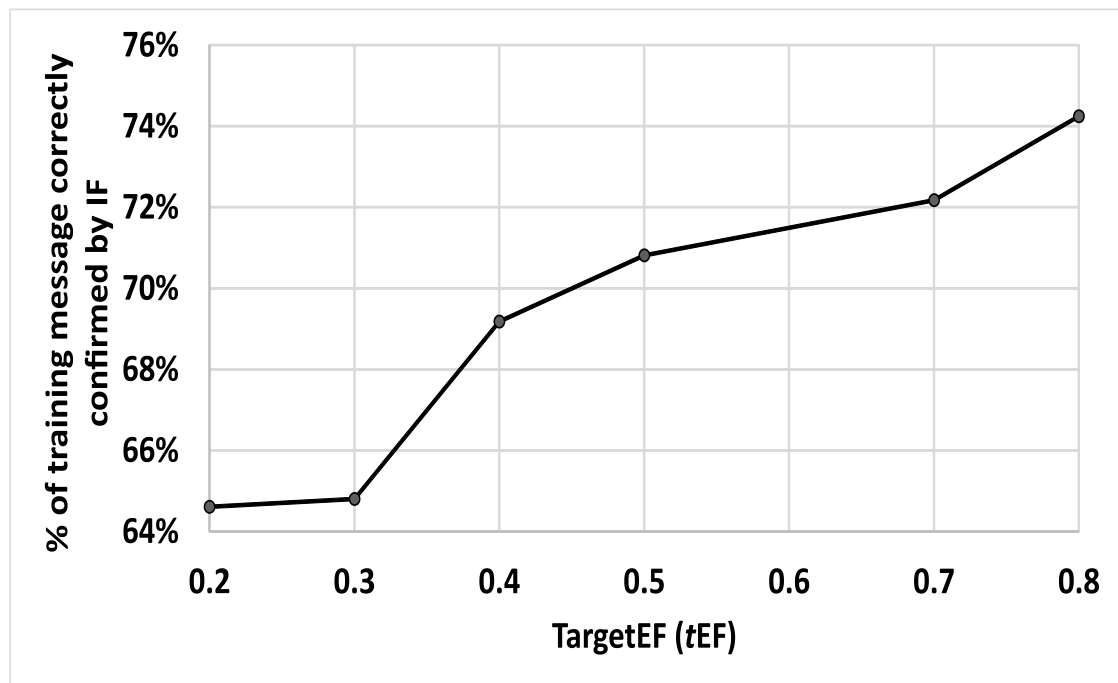


Figure 4.7: Percentage of training messages correctly confirmed by IF for different levels of TargetEF.

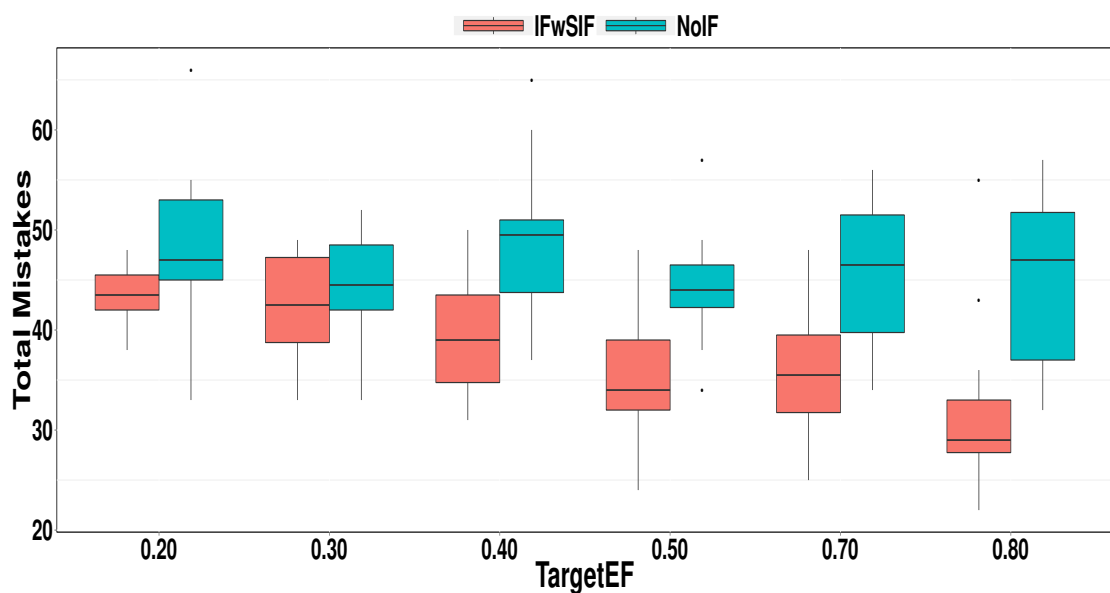


Figure 4.8: Total mistakes for different levels of TargetEF.  $p$ -value  $< 0.05$  for a two-sided Welch's two sample  $t$ -test suggests that we have sufficient evidence to conclude that the total number of mistakes in NoIF is greater than the number of mistakes in IFwSIF.

64% of the confirmed messages have correct tags, whereas at a Target EF of 0.80 this has improved to 74%. In all cases, this shows that the training examples created by IFwSIF contain a lot of noise in the target tags. We were pleasantly surprised to see that on balance the classifier still benefited from these noisy training examples. This was particularly surprising because of the well-known vulnerability of the confidence weighted classifier to mislabeled training examples.

Finally, to test the statistical significance of the error reduction, Figure 4.8 displays box plots of the total mistakes on the last 70 messages for NoIF and IFwSIF. At each level of TargetEF and for all levels combined, the differences between the two algorithms are statistically significant at  $p < 0.05$ . This provides strong evidence that IFwSIF is giving a real improvement over NoIF.

## Chapter 5: Knowledge Worker Case Study

In this chapter we describe the case studies we conducted with the real users of the TAPE system: my advisor, Thomas Dietterich and myself. We both are long term users of TAPE on our email. We accumulated our email data for several months, and then performed the implicit feedback experiments via the TAPE replay mode. The results from these case studies show the behavior of the system and the impact of the implicit feedback models on real knowledge workers' regular usage on the system.

### 5.1 Case Study 1: A Graduate Student

I have been using the TAPE email predictor for over 2 years now. In this section, I describe how I accumulated the data set for the implicit feedback experiments and then analyze the results.

#### 5.1.1 The Data Set

I have accumulated a total of 4982 email messages over a period of ten months (we will call this data set 'Sorower-Data'). On average, I received 17 messages per day, and over time, I have defined a total of 43 tags to categorize these messages. The number of ground truth tags per message is 1.23. Some of the tags are research project related, some are meetings or travel related, and some are about my personal finance or health. Figure 5.1 shows the distribution of tags. Most of the tags appear only a few times, whereas some tags appear more frequently. Examples of some of the most frequent tags are MLPostings/Jobs – forum messages that include jobs postings; TaskTracer – messages related to one of my active research projects; OSU\_EECS – administrative email messages from the school of EECS. Examples of some of the rarely used tags are MLTools/Data – messages related to machine learning tools or data; News – messages that contain news and do not fall under any other tag; Event – messages about events that do not fall under any other tag.

During the course of this data collection, all of the implicit feedback activities were

recorded. Figure 5.2 shows the total number of implicit feedback events recorded, plotted in a log 10 scale in order to fit the wide range of values within a single plot. ‘Message R/O’ indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector, and hence, ‘Message R/O’ has the highest frequency. Note that we excluded the copy, move to folder and print events, because the data set Sorower-Data didn’t have any instance of these events.

32% of these messages were reserved for training (I started collecting implicit feedback data after training on these messages), 20% for validation and tuning of the parameters, and the remaining were reserved for testing.

For evaluation, we first train TAPE on the training messages (exactly as I had the system trained). Then the test messages and the corresponding events are processed by TAPE via the “replay mode” using each of the implicit feedback algorithms.

### 5.1.2 Parameter Learning

First, we learn the parameters of the self training and implicit feedback models using the validation data set.

For self training (STrain), we must learn two parameters: positive prediction training lower threshold ( $PL$ ) and negative prediction training upper threshold ( $NU$ ). After predicting tags on a message, STrain creates a positive training example for each tag that has a predicted probability greater than  $PL$ , and creates a negative example each tag that has a predicted probability of being positive of less than  $NU$ . We apply a script that varies  $PL$  and  $NU$  on the validation data set, and to find the parameter values that minimize total mistakes.

Figure 5.3 shows the total mistakes computed over the validation data set, plotted as a function of  $PL$  and  $NU$ . Note that the values  $PL = 1.0$  and  $NU = 0.0$  cause there to be no self training. As we start decreasing  $PL$ , there are opportunities for training on predicted positive tags. Although some of these high confident tags can be incorrect, we would expect most of them to be correct. Hence, training on them improves the performance of the system. However, as we continue decreasing  $PL$ , we allow training on more false positives, and this causes the system to perform worse. Similarly, as we start increasing  $NU$ , the system initially performs better but the gain in performance diminishes as we increase  $NU$  even more. As shown in Figure 5.3, the optimal values for

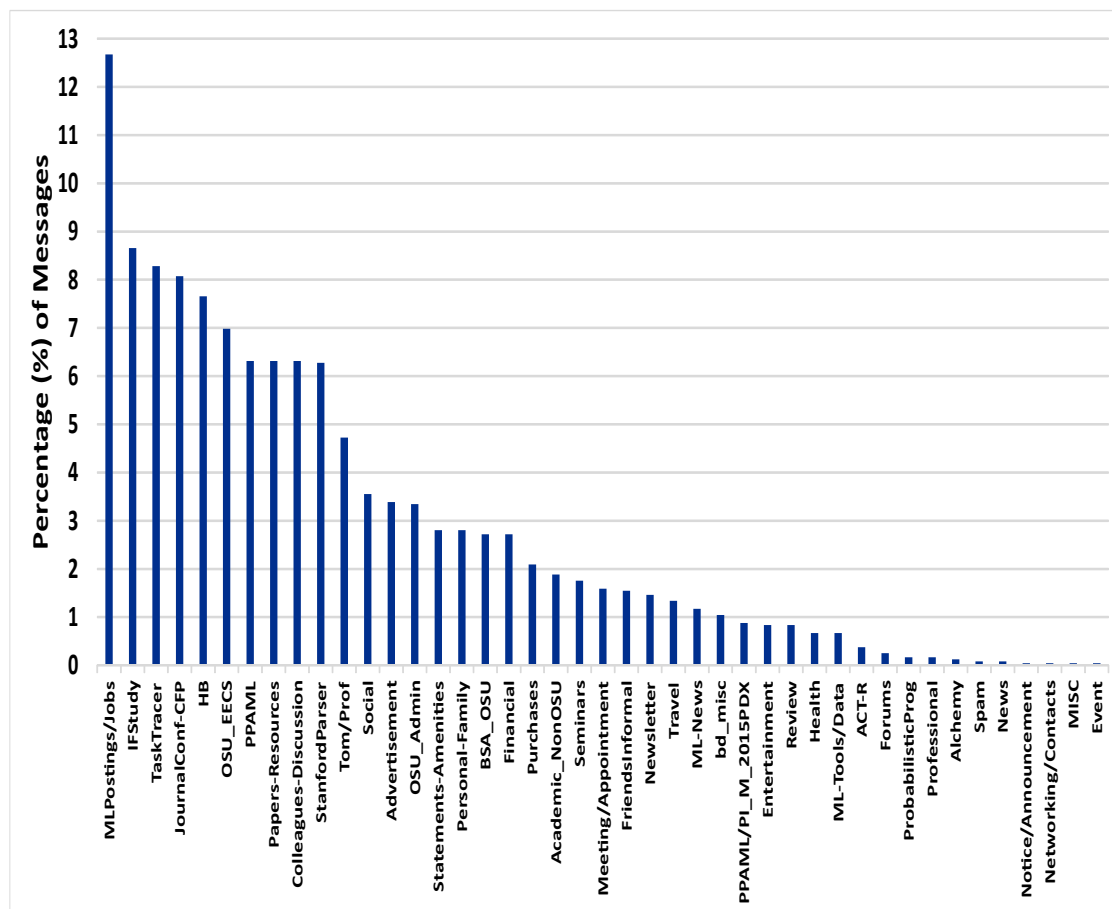


Figure 5.1: Distribution of Tags in Sorower-Data.

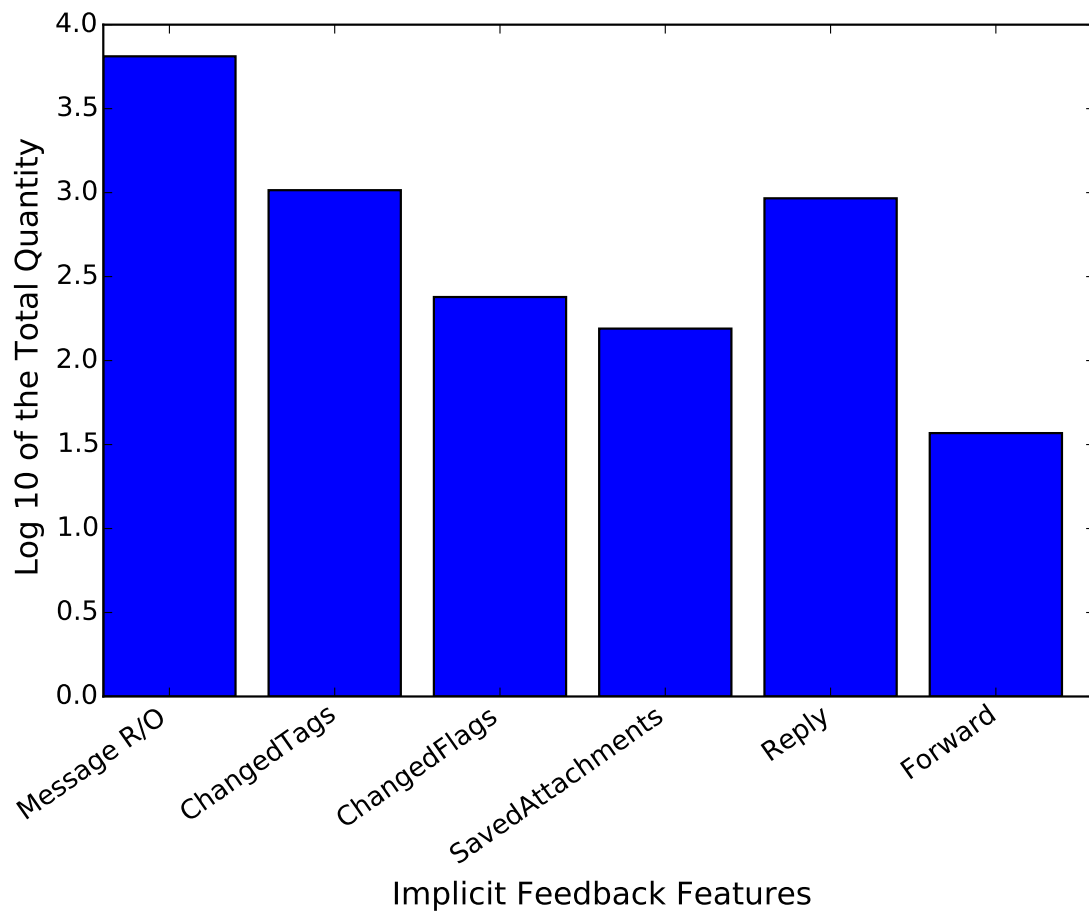


Figure 5.2: Implicit feedback events captured in Sorower-Data plotted on a log 10 scale. 'Message R/O' indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector.



Sorower-Data are  $PL = 0.95$  and  $NU = 0.05$ .

We have shown with the user study data that there exists a threshold on the total number of implicit feedback actions such that the loss in performance by training on incorrect examples (but confirmed by IF) is out-weighed by the gain of training on correctly confirmed examples (Figure 4.4). To learn the IF threshold on Sorower-Data, we follow the same approach of using the validation data set. In Figure 5.4, we plot the number of bad and good training examples and the final cumulative number of mistakes as a function of the threshold. The minimum point of the final cumulative mistakes curve shows that ‘IF threshold = 4.0’ is the optimal value for Sorower-Data.

Finally, we learn the weights on the implicit feedback events so that we can evaluate the IFwSIFLW method. Recall that we have the implicit feedback events recorded over time for Sorower-Data. Because these are my personal messages, I was able to manually inspect each message in the validation data set and to confirm that the ground truth tags were correct. This gives us the training data for solving the optimization problem described in Equations 3.3–3.8. We solve this using CVX, a package for specifying and solving convex programs [16, 17]. Figure 5.5 shows the learned weights as a bar plot. Note that the longer reading time is an important indicator of confirmation of predicted tags. I frequently use flags on messages that require further attention, and while doing so, often I fix incorrect tags, if there are any. That is why the ‘change flag’ event is a good indicator of my attention to tags, while forwarding an email and saving an attachment contribute only a little.

### 5.1.3 Results Analysis

In Chapter 4, we have already shown that in a lab-controlled user study, implicit feedback based models improve the performance of the TAPE email predictor. Now that we have learned the parameters of all of our models for Sorower-Data, Figure 5.6 shows the cumulative prediction mistakes of each of the seven algorithms. As expected, the complete online feedback model accumulates the smallest number of errors. The NoIF model accumulates more errors than the IF-based models, and IFwSIF is able to eliminate the gap between NoIF and Online models by about 50%. The SIF algorithm, by training as soon as the user changes any one tag, is the second best IF model. IFwoSIF makes more mistakes than SIF but that is still better than no implicit feedback at all (NoIF).

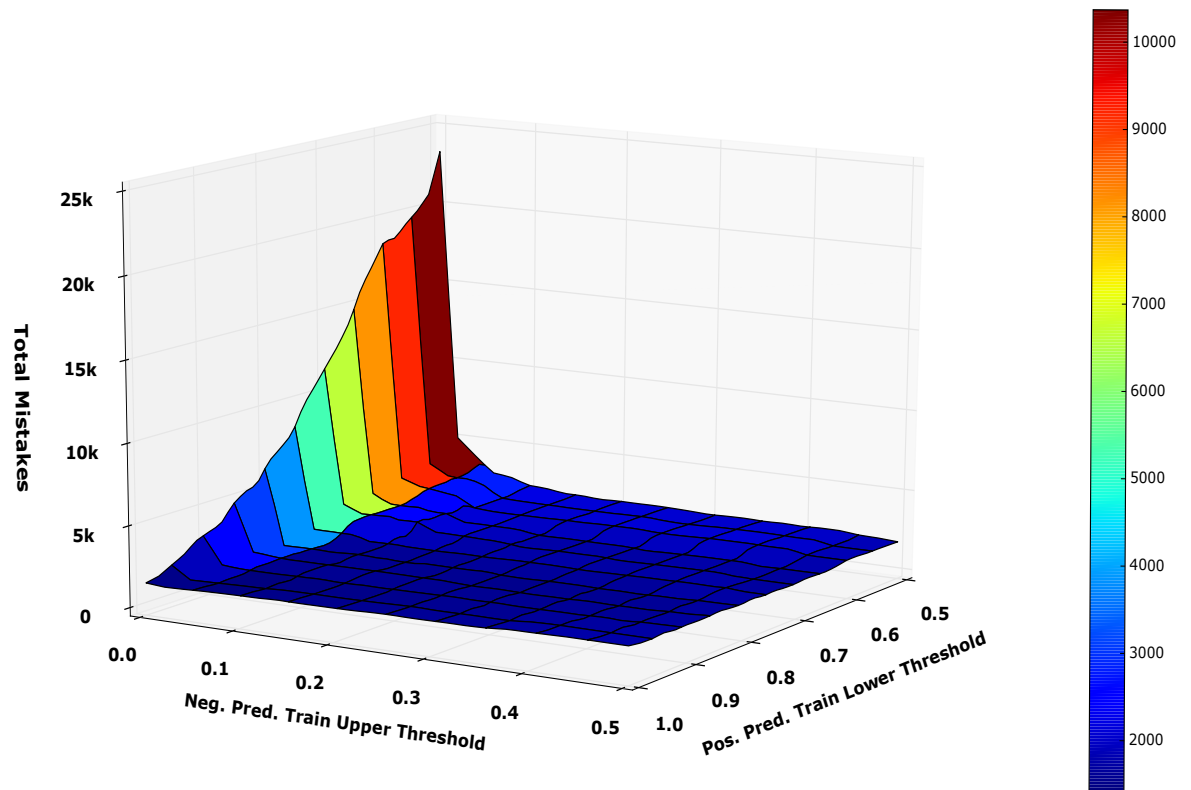


Figure 5.3: Total mistakes made by self-training on Sorower-Data, plotted as a function of ‘Positive Prediction Training Lower Threshold’ and ‘Negative Prediction Training Upper Threshold’.

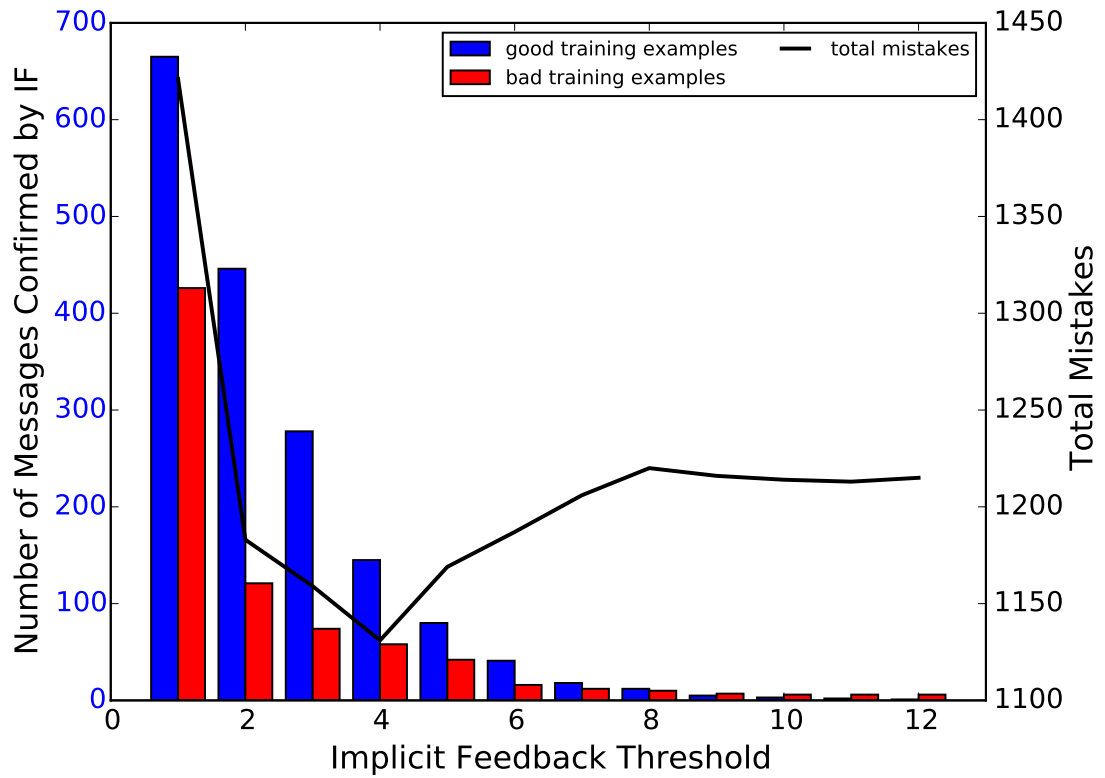


Figure 5.4: Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold, computed on Sorower-Data.

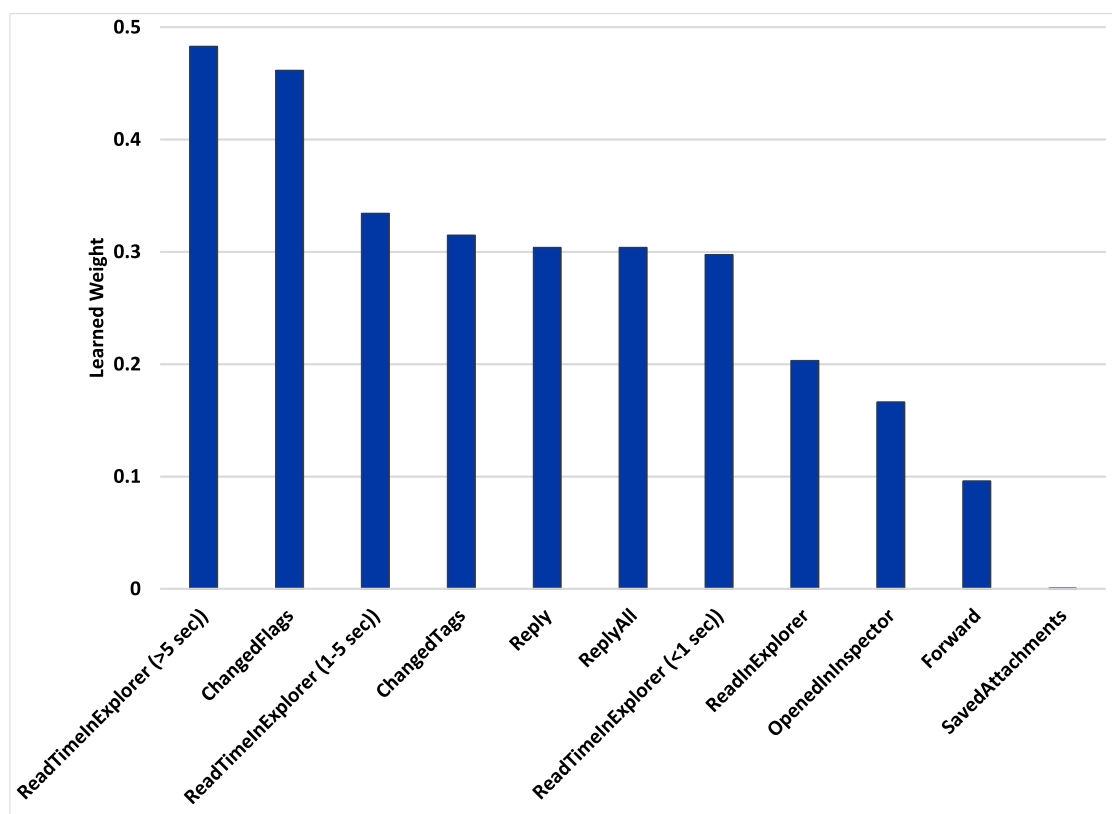


Figure 5.5: Weights of the implicit feedback events learned on Sorower-Data.

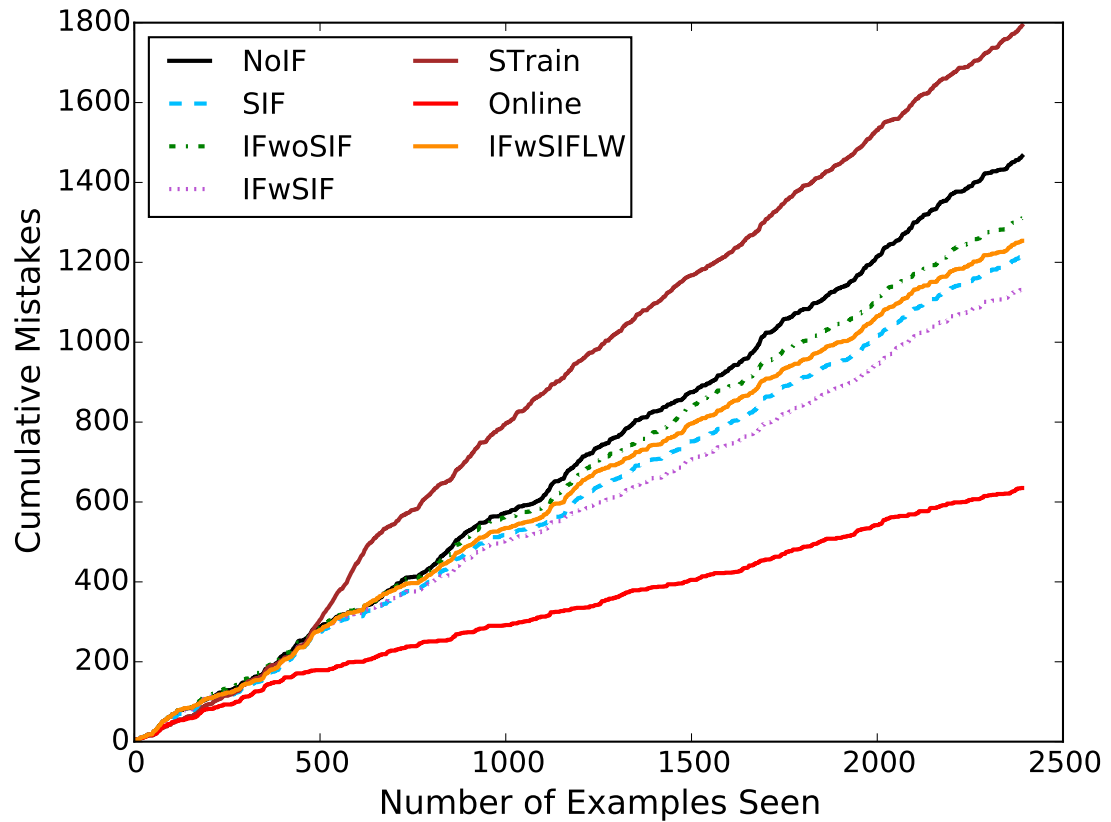


Figure 5.6: A comparison of the cumulative mistakes of each of the seven IF algorithms on Sorower-Data.

This matches our findings with the user study data. As we have seen with the user study data, this shows that simple implicit feedback and implicit feedback (i.e., training when the number of implicit feedback events exceeds a threshold) both provide good training examples, and combining them using IFwSIF gives better results than either method alone.

As we described above, the benefit of implicit feedback is due to the fact that it allows additional training over the NoIF model. Although implicit feedback is noisy and there may be some training on incorrect tags, the overall benefit is outweighed towards training on correct tags, and hence, the gain in performance. The total number of positive training examples and the total number of negative training examples produced by each

of the models are shown in Figure 5.7 and 5.8. The online model receives the highest number of positive and negative trainings, and in contrast the NoIF model receives the lowest number of positive and negative training. This explains why online and NoIF are the best and the worst performers respectively. IFwoSIF receives more training (both positive and negative) than NoIF, and SIF receives more training than IFwoSIF. Clearly, combining IFwoSIF and SIF achieves far more training, and that is why IFwSIF is the best of all the IF models.

Although one would expect IFwSIFLW (IFwSIF with learned weights) to perform better than or equal to IFwSIF, Figure 5.6 shows the both IFwSIF and SIF performed better than IFwSIFLW. Figure 5.7 and 5.8 shows that IFwSIFLW generated fewer training examples than IFwSIF. This may be for two reasons. First, in the current version of TAPE, the implicit feedback events are recorded without the exact time stamp at which they occur. Therefore, the data had to be preprocessed before it could be used to solve the convex optimization problem, and this process was not perfect. Second, the convex optimization equations has three trade-off parameters ( $C_N, C_P$  and  $C_b$ ). More experimental data and more exploration is necessary to tune these parameters to find the optimal values.

The biggest difference between the user study and the results on Sorower-Data was the performance on the self training model, STrain. Figures 5.7–5.9 shows that STrain creates many positive and negative training examples, more than any of the implicit feedback models, and nearly in the range of the number of training examples that the online model creates. Despite that, STrain accumulates the largest number of errors. This suggests that, although STrain creates a lot of training examples, many of these examples must be incorrect. To explore this further, we compute how many additional training examples IFwSIF and STrain models create as compared to the NoIF model, and how many of these examples are actually correct (e.g., the message has only correct tags). In Figure 5.10, we compare the additional number of positive training examples that IFwSIF and STrain create for the data set Sorower-Data. IFwSIF creates only 250 additional training examples, and many (=179) of these are good training examples (e.g., messages with correct tag). In contrast, STrain creates about 8 times more additional positive training examples but it also creates 7 times more bad training examples. Although in the short period of the user study data, the consequence may be acceptable (e.g., better than NoIF), clearly, in the long run, training on a large number of bad examples causes the

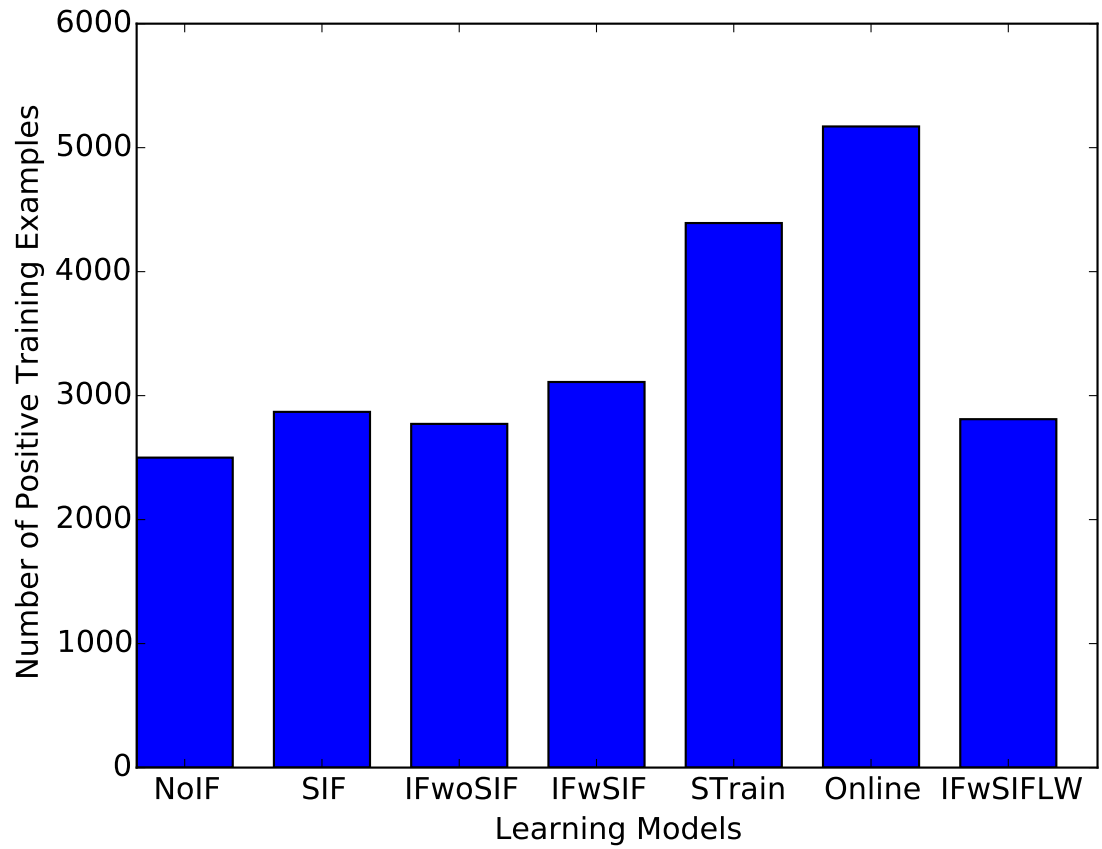


Figure 5.7: Total number of positive training examples on Sorower-Data.

STrain model to perform the worst.

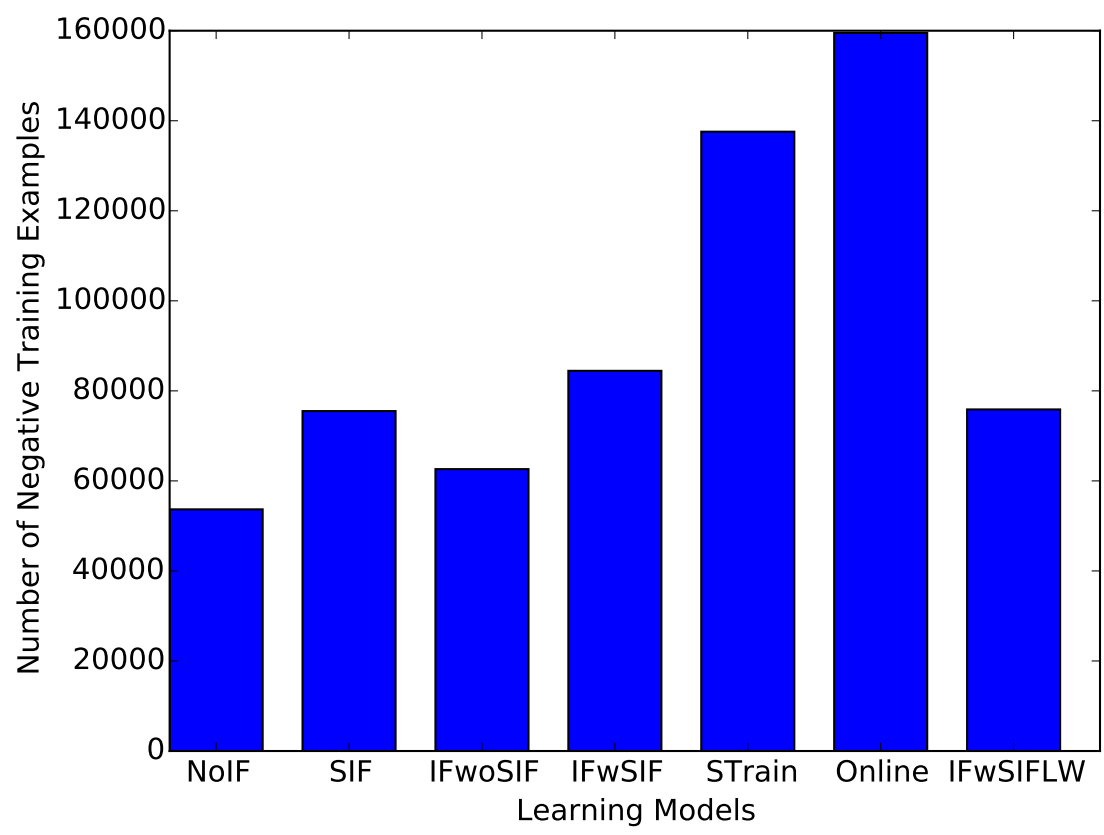


Figure 5.8: Total number of negative training examples on Sorower-Data.



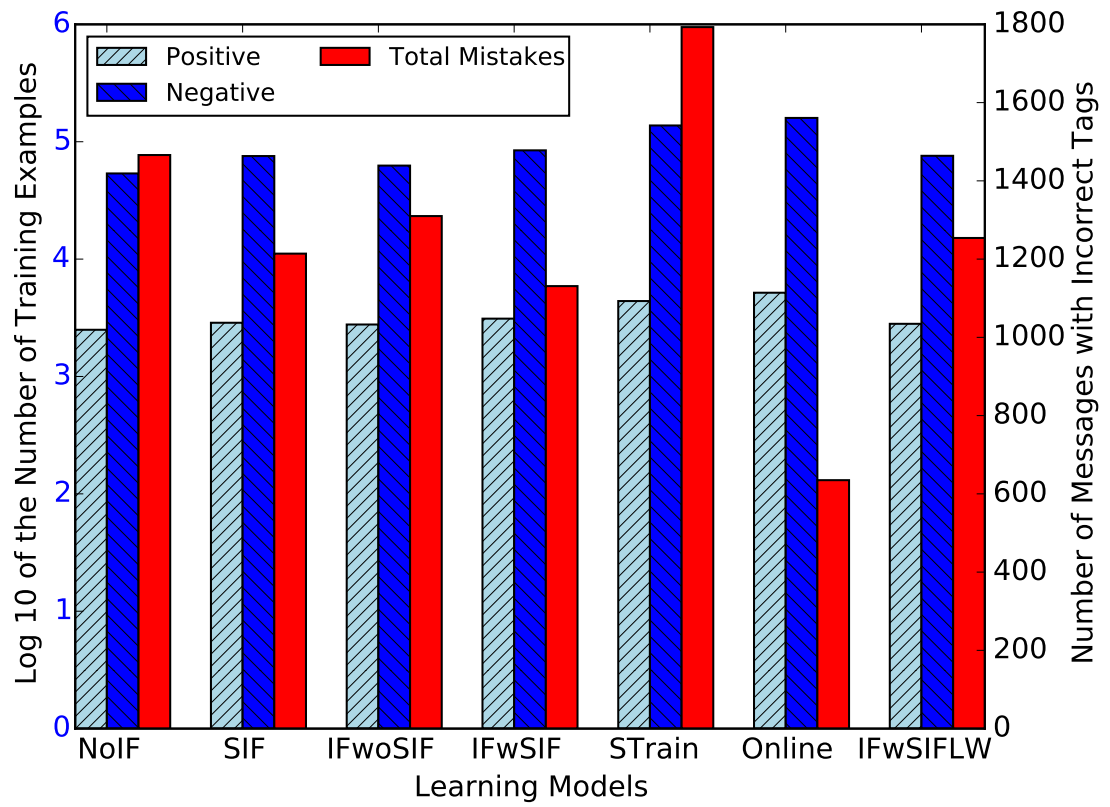


Figure 5.9: Total number of training examples (left axis) and total number of prediction mistakes on Sorower-Data.

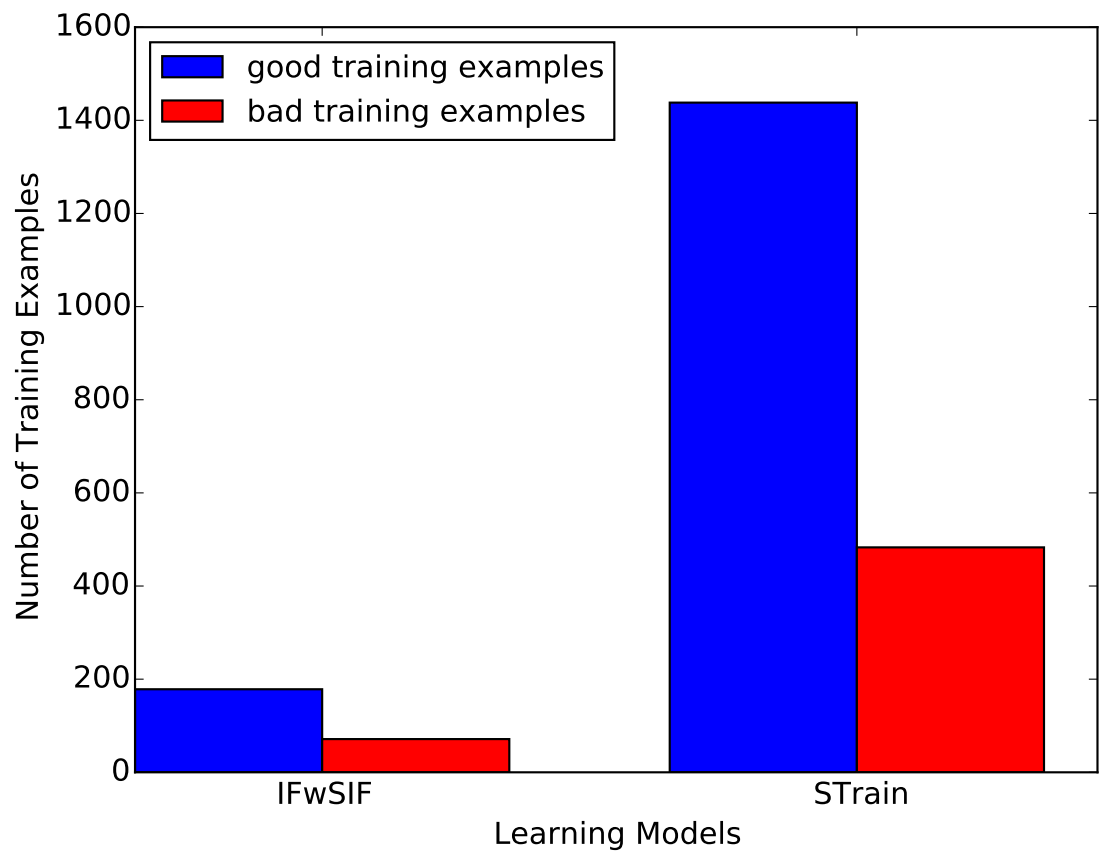


Figure 5.10: Total additional training examples created by IFwSIF and STrain on Sorower-Data.

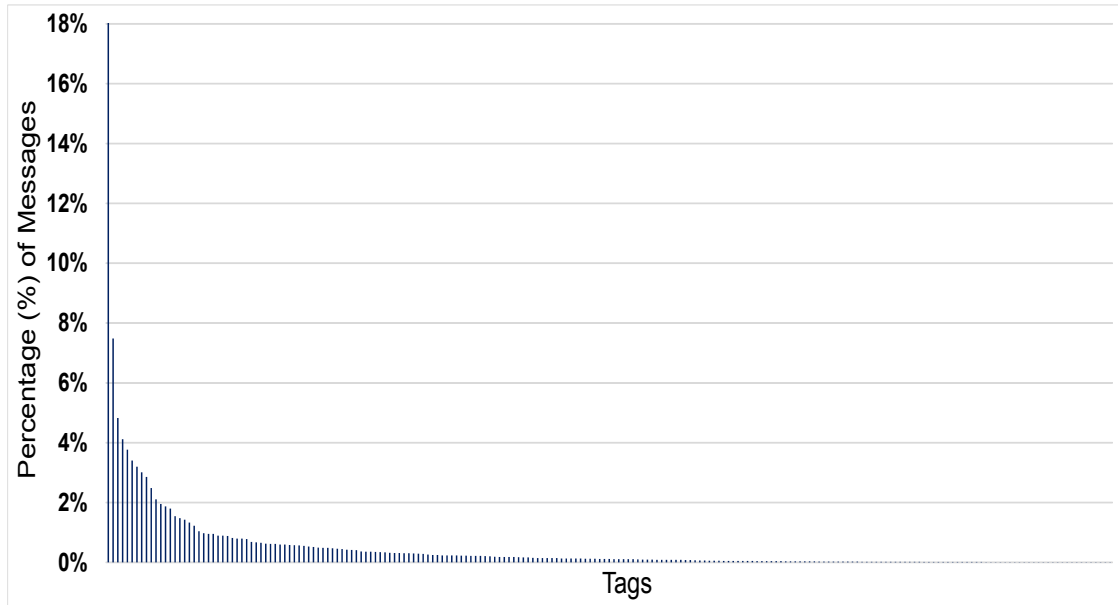


Figure 5.11: Distribution of Tags in TGD-Data.

## 5.2 Case Study 2: A Professor

Thomas (Tom) Dietterich has been an active user of the TAPE system for the past several years. In this section, we describe the his email data set (we call it ‘TGD-Data’) that we used for the implicit feedback experiments and then analyze the results.

### 5.2.1 The Data Set

Tom accumulated a total of 13721 of his own email messages over a period of approximately six months. On average, he received 66 messages per day, and over time, he defined a total of 214 tags to categorize these messages. The number of ground truth tags per message is 1.07. Figure 5.11 shows the distribution of tags <sup>1</sup>.

During the course of this data collection, all of the implicit feedback activities were recorded. Figure 5.12 shows the total number of implicit feedback events recorded – plotted on a log 10 scale in order to fit the wide range of values within a single plot.

<sup>1</sup>the exact tag-names in TGD-Data have been omitted to protect the privacy of the data set owner, Thomas Dietterich.

‘Message R/O’ indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector, and hence, ‘Message R/O’ has the highest frequency. Note that we excluded the changed flag, copy, move to folder and print events because the data set TGD-Data didn’t have any instance of these events.

73% of these messages were reserved for training (Tom started collecting implicit feedback data after training on these messages), 7% for validation and tuning of the parameters, and the remaining were reserved for testing.

For evaluation, we first train TAPE on the training messages (exactly as Tom had the system trained). Then the test messages and the corresponding events are processed by TAPE via the “replay mode” using each of the implicit feedback algorithms.

## 5.2.2 Parameter Learning

To learn the parameters of the self training (STrain) and implicit feedback models using the validation data set, we follow the same steps as we did with Sorower-Data (Section 5.1.2). Recall that, in the STrain model we have to learn two threshold parameters: positive prediction training lower threshold ( $PL$ ) and negative prediction training upper threshold ( $NU$ ). We vary  $PL$  and  $NU$  on the validation data set in TGD-Data and find the best set of parameters that minimizes total mistakes.

Figure 5.13 shows the total mistakes computed over the validation data set, plotted as a function of  $PL$  and  $NU$ . Recall that  $PL = 1.0$  and  $NU = 0.0$  imply that there is no self training. As we decrease  $PL$ , the system trains on more and more predicted positive tags, and there is an optimal point in  $PL$  where the total mistakes is minimized. For TGD-Data,  $PL = 0.95$  is this optimal point (this is the same as in Sorower-Data).

We expected the same thing to happen as we increase  $NU$  because this allows training on more predicted negatives. Surprisingly, the optimal value for  $NU$  is at 0.50, which is much higher than what we learned in case of Sorower-Data ( $NU = 0.05$ ). To investigate this further, we present another view of this total mistakes plot (Figure 5.14). We plotted  $PL$  on the right axis. For each value of  $PL$ , we vary  $NU$  between 0.0 and 0.50, plotted on the horizontal axis. Finally, on the left axis, we plot total mistakes as a bar chart that shows that for any given level of  $PL$ , as we train on more negatives (increase  $NU$ ), the total amount of mistakes generally decrease.

For a fixed value of  $PL$ , we allow training on more negative predictions as we increase

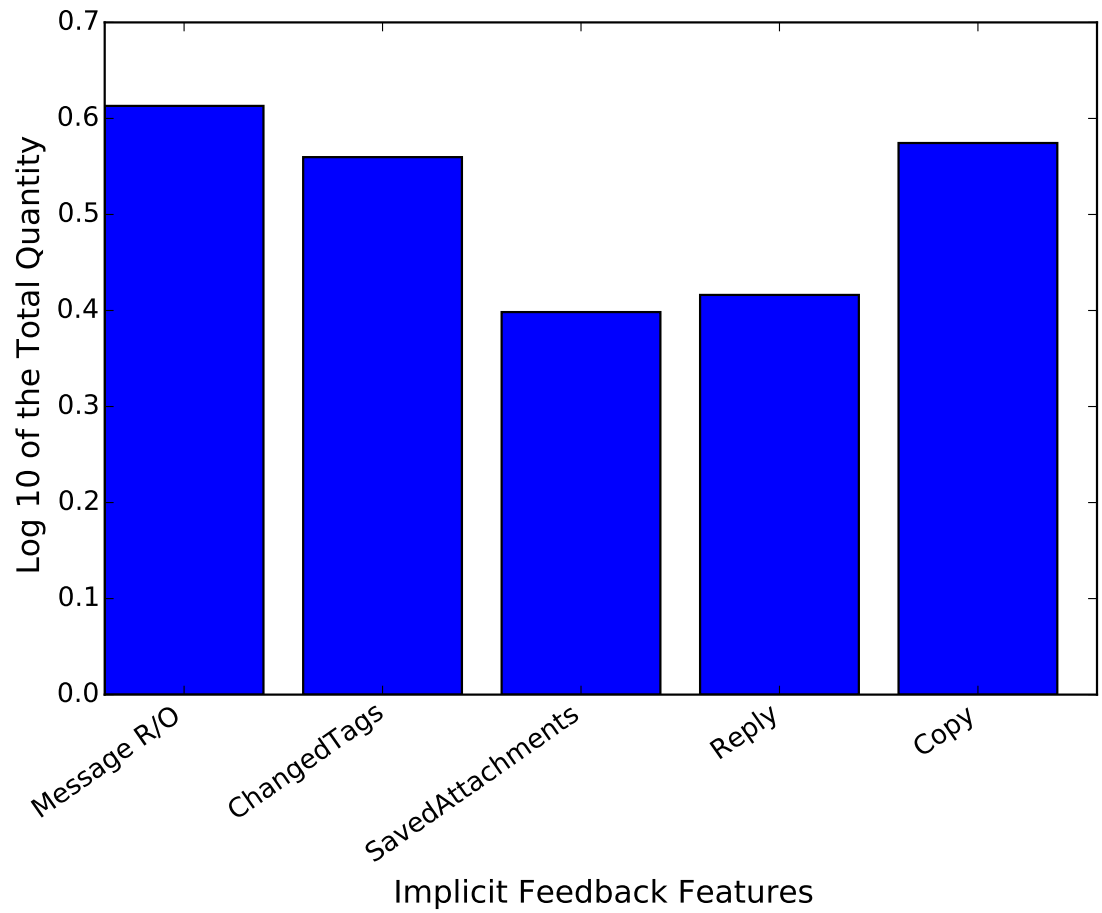


Figure 5.12: Implicit feedback events captured in TGD-Data plotted in log 10 scale. 'Message R/O' indicates the total number of times the message was opened in Outlook Explorer or in Outlook Inspector.

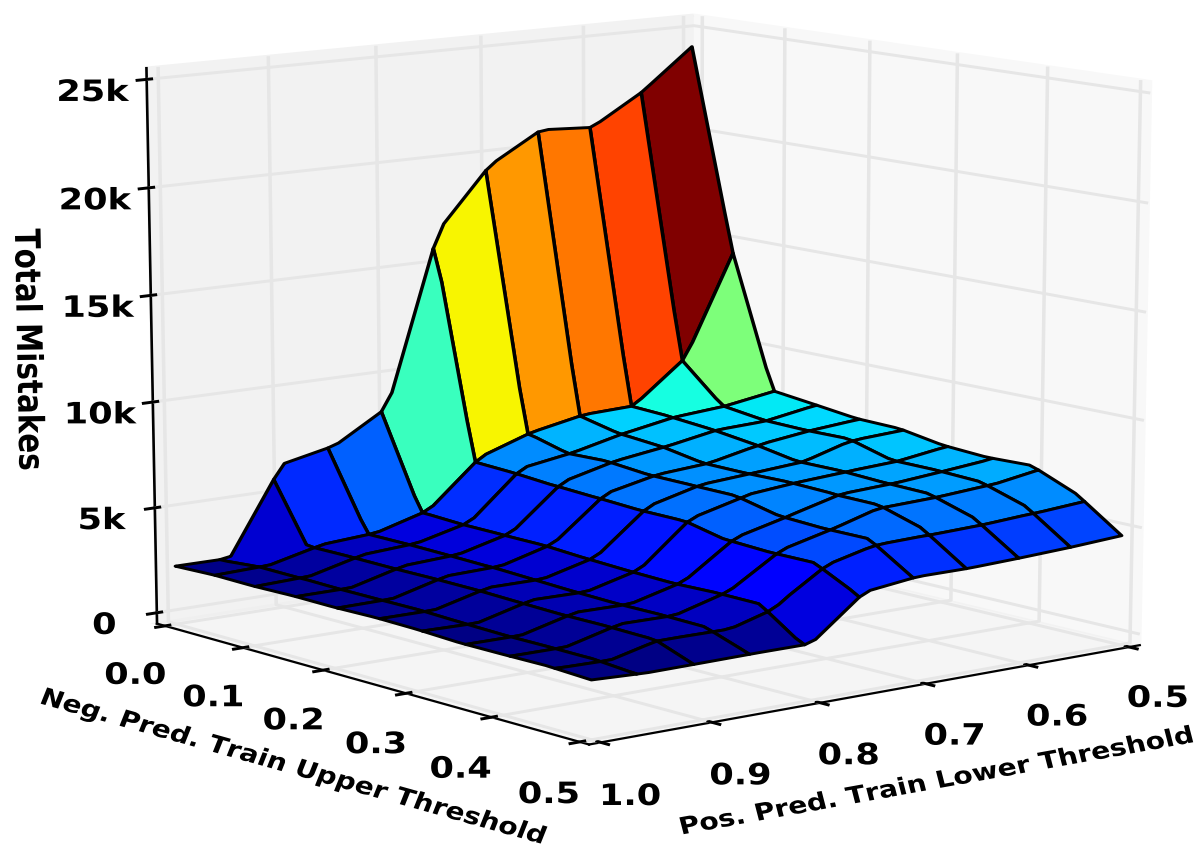


Figure 5.13: Total mistakes made by self-training on TGD-Data, plotted as a function of ‘Positive Prediction Training Lower Threshold’ and ‘Negative Prediction Training Upper Threshold’.

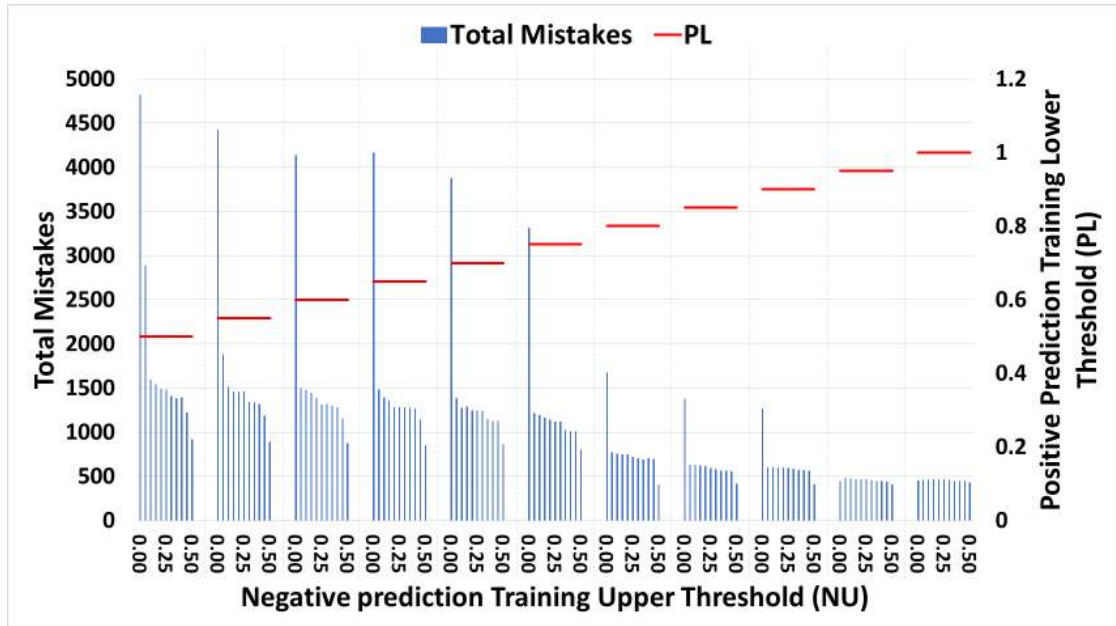


Figure 5.14: Total mistakes made by self-training on TGD-Data plotted as a bar plot (left axis) to show the effects ‘Positive Prediction Training Lower Threshold’ (right axis) and ‘Negative Prediction Training Upper Threshold’ (horizontal axis).

$NU$ . Some of these training decisions are incorrect i.e., negative training on a tag that is predicted with lower probability but was promoted as predicted tag, and the user didn’t correct it (and hence, is the correct tag for the message). However, many of these training decisions are indeed correct because for a particular message, most of the tags are actually negative tags. Especially in case of TGD-Data, there are 214 user-defined tags and the average number of tags per message is 1.07. This means, for most messages there is only one correct tag and the remaining 213 are incorrect tags. Therefore, with the increase of  $NU$ , we will generally be training on negatives that are correct, and only a few that are incorrect. This explains the improvement in performance as we increase  $NU$  for a fixed value of  $PL$  in Figure 5.14.

For the rest of the experiments with TGD-Data, we set  $PL$  and  $NU$  to be 0.95 and 0.50 respectively that was learned using the validation data.

To learn the IF threshold on TGD-Data, we follow the same approach using the validation data set. In figure 5.15, we plot the number of bad and good training examples

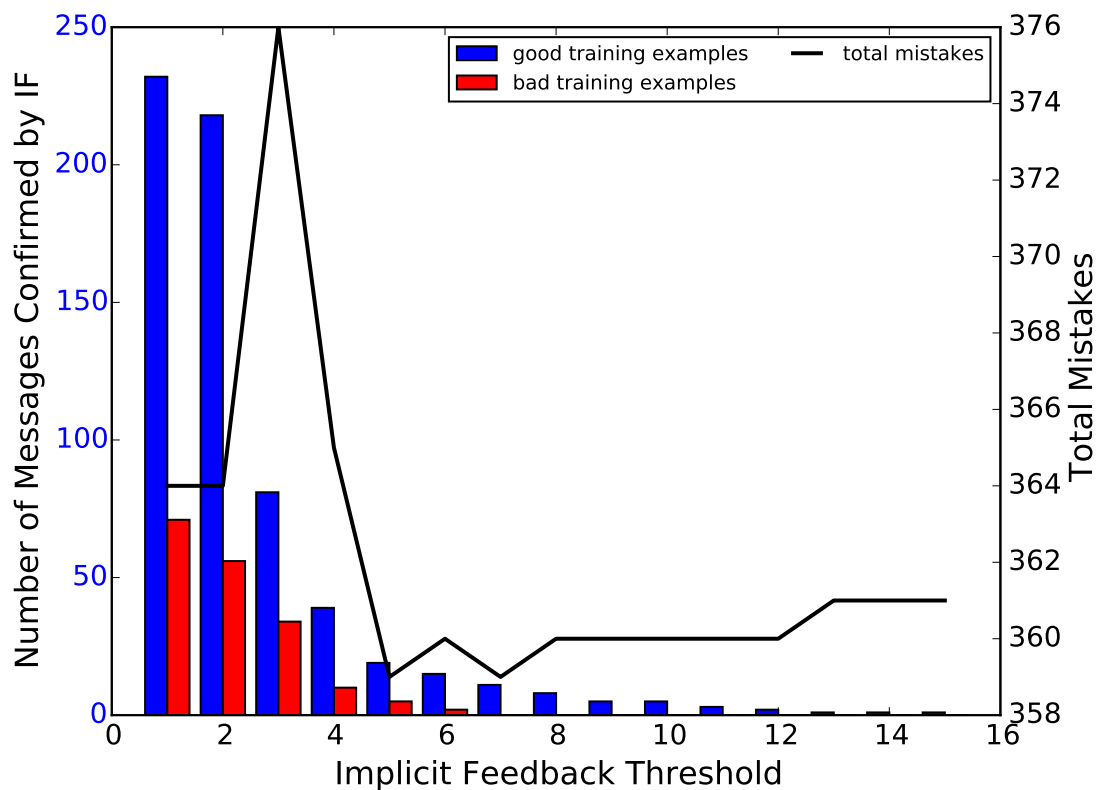


Figure 5.15: Total mistakes (right axis), total number of good and bad training examples (left axis) created by IFwSIF for different levels of the implicit feedback threshold, computed on TGD-Data.



and the final cumulative number of mistakes as a function of the threshold. The minimum point of the final cumulative mistakes curve shows that ‘IF threshold = 5.0’ is the optimal value for TGD-Data.

We did not test IFwSIFLW (learn weights) on the implicit feedback event for TGD-Data. Learning the weights on the implicit feedback events require extensive ground truth labeling, preprocessing, and preparing the data for the optimization solver. We didn’t have the ground truth labeling for this larger data set.

### 5.2.3 Results Analysis

Figure 5.16 shows the cumulative prediction mistakes for all of the algorithms except for IFwSIFLW. This shows a result very similar to what we have seen with Sorower-Data and with the user study data. The complete online feedback model accumulates the smallest number of errors. The NoIF model accumulates more errors than the IF-based models, and IFwSIF is able to eliminate the gap between NoIF and Online models by more than 50%. The SIF algorithm is the second best IF model followed by IFwoSIF, which is still better than no implicit feedback at all (NoIF).

One interesting thing to note in this plot is that for about first 2000 messages, IFwSIF and SIF perform nearly identically. This is likely because SIF largely dominated up to this point, and the additional benefit of implicit feedback events is only evident after that.

The total numbers of positive training examples and negative training examples produced by each of the models are shown in Figure 5.17 and 5.18. As in all previous experiments, the online model generates the highest number of positive and negative training examples, and in contrast the NoIF model generates the lowest number of positive and negative training examples. This explains why the online and the NoIF models are the best and the worst respectively. Among all the IF models, combining IFwoSIF and SIF (IFwSIF) generates the largest amount of training, and that is why IFwSIF is the best of all the IF models.

The self training model, STrain, again performs worse than the NoIF model. Figures 5.17–5.19 shows that STrain creates many positive and negative training examples, more than any of the implicit feedback models, and sometimes even more than online (figure 5.17). Despite that, STrain accumulates the largest number of errors. To investigate

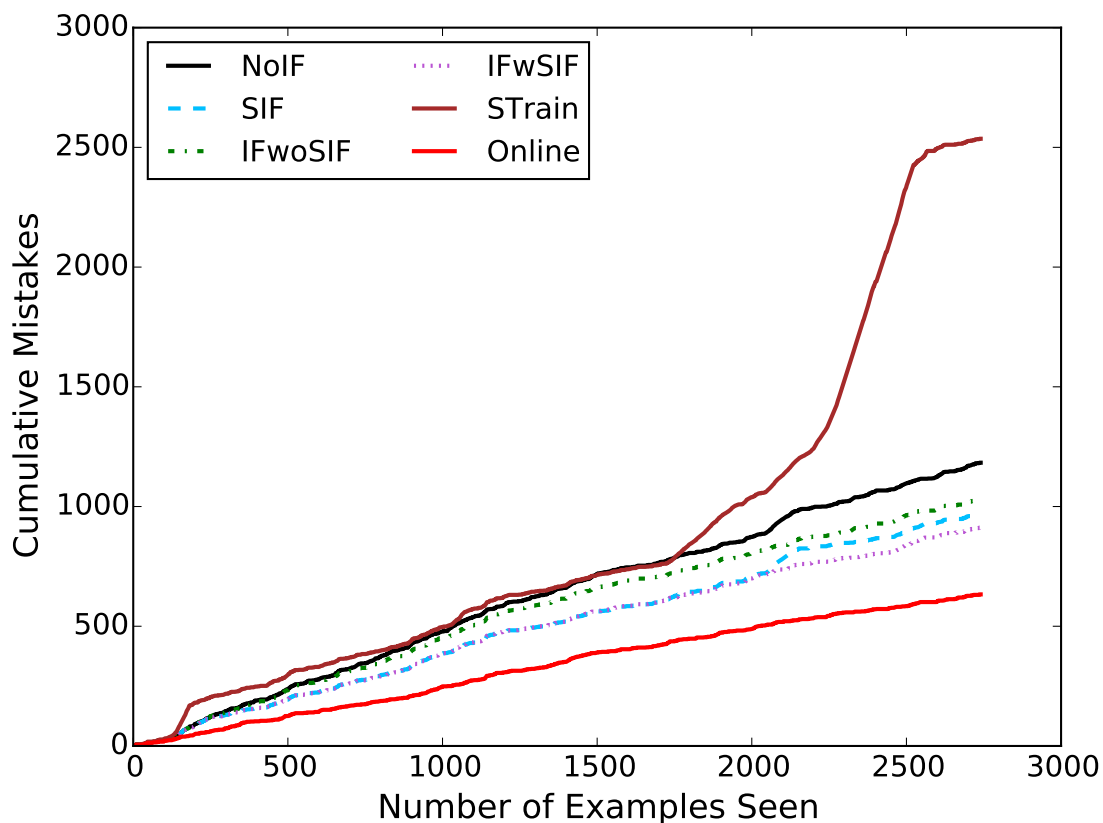


Figure 5.16: A comparison of the cumulative mistakes of each of the seven IF algorithms on TGD-Data.

this further, we compute how many additional training examples IFwSIF and STrain models create as compared to the NoIF model, and how many of these examples are actually correct (e.g., the message has only correct tags), similar to what we did with Sorower-Data. In Figure 5.20, we compare the additional number of positive training examples that IFwSIF and STrain create for the data set TGD-Data. IFwSIF creates only 114 additional training examples, and many (=86) of these are actually good training examples (e.g., messages with correct tag). In contrast, STrain creates about 30 times more additional positive training examples, but about 45% of these are bad training examples. Training on a large number of bad examples causes the STrain model to perform the worst.

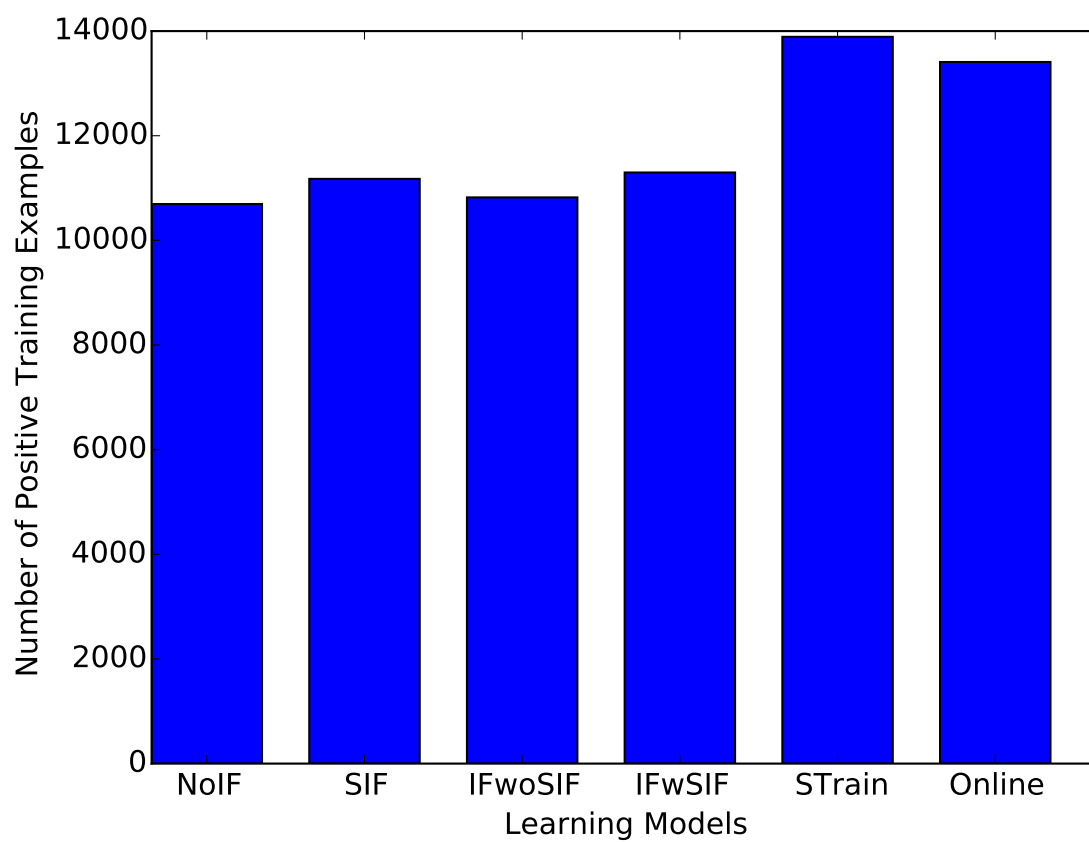


Figure 5.17: Total number of positive training examples on TGD-Data.

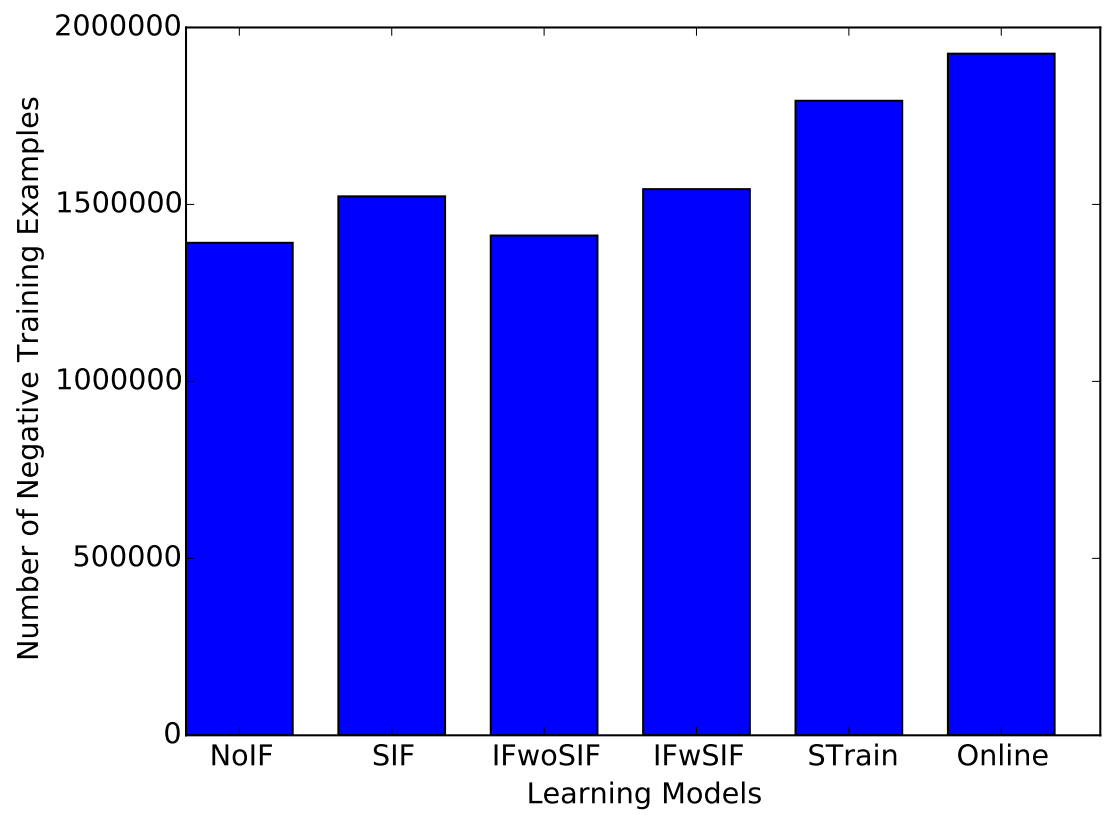


Figure 5.18: Total number of negative training examples on TGD-Data.

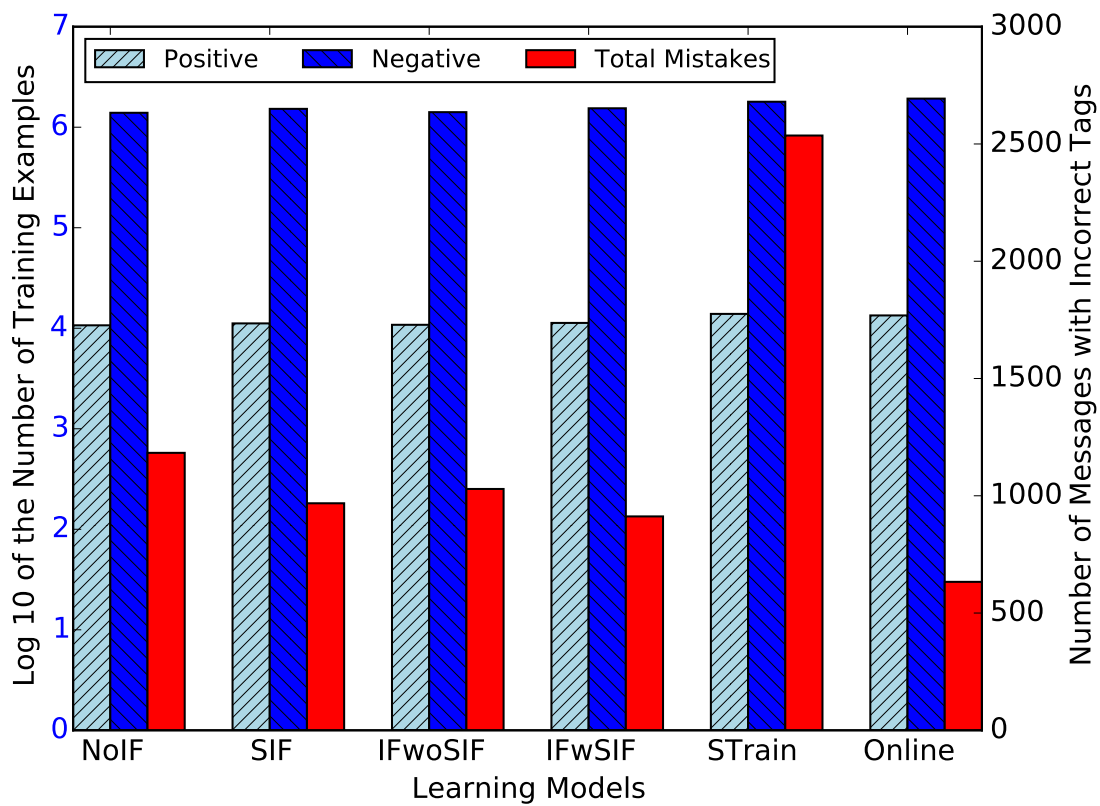


Figure 5.19: Total number of training examples (left axis) and total number of prediction mistakes on TGD-Data.

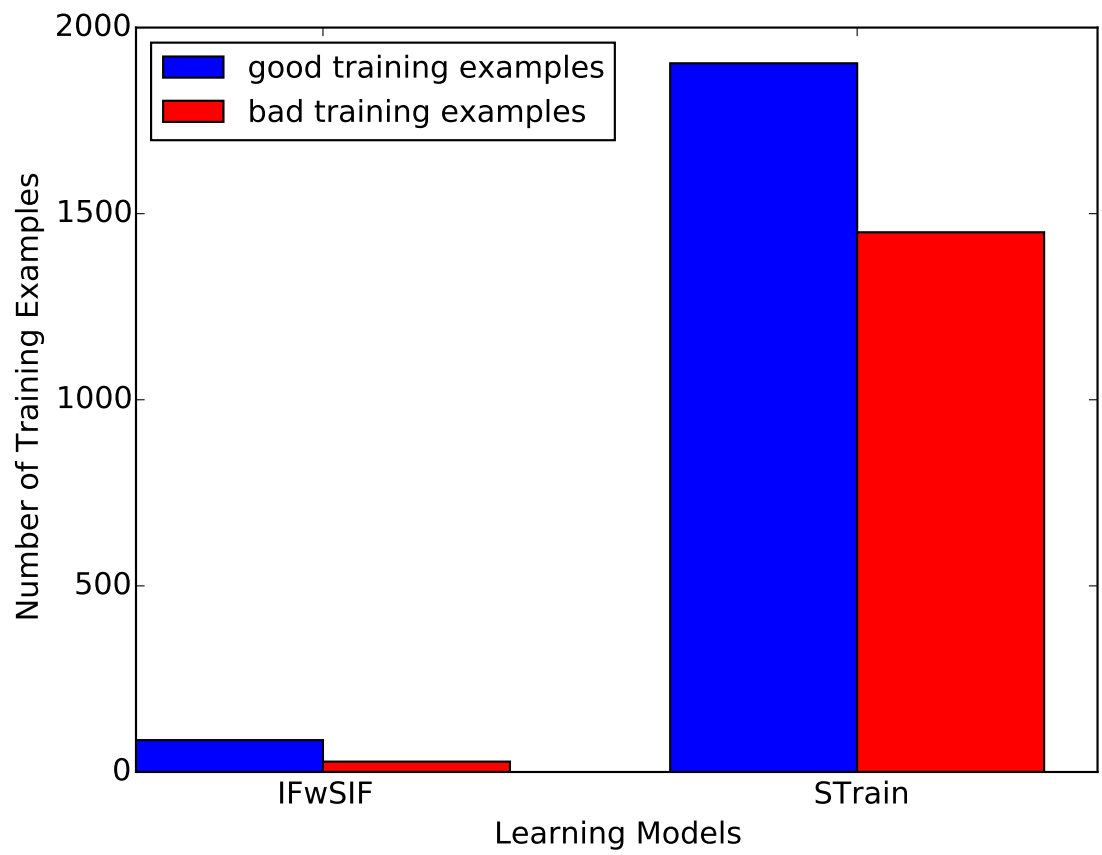


Figure 5.20: Total additional training examples created by IFwSIF and STrain on TGD-Data.

### 5.3 Summary

Table 5.1 summarizes our case study (and the user study) data sets and the results we have from our experiments as described in the above sections. We had only 330 messages in our lab-controlled user study and the users were constrained to 6 predefined tags only (users were not allowed to create new tags). In contrast, both of our case study subjects have much larger data sets and there was no restriction against creating new tags. TGD-Data has about 3 times more messages and about 4 times more tags than Sorower-Data. Most of the messages in TGD-Data are likely to have one correct tag because the label cardinality of this data set is 1.07. The label cardinalities of the user study data set and Sorower-Data is 1.24 and 1.23 respectively. Table 5.1 summarizes the detailed properties of these data sets.

Learning the parameters for the self training (STrain) model requires the validation and parameter tuning experiment to be run for many values for  $PL$  and  $NU$ . Since our user study data was very small, we could not allocate a validation data set to learn the STrain parameters. Instead, we choose the threshold values that reduced the total mistakes for a few randomly selected participants on their simulated data for several levels of targetEF. We utilized the portions of the case study data sets reserved for validation to learn  $PL$  and  $NU$  thresholds. The threshold values we learned for the two case studies are slightly different from each other and are very different from the values we learned for the user study data sets.

The implicit feedback threshold we learn for the user study data set is slightly higher than the thresholds we learn with the case study data sets. This is because the user study data set was designed such that the users often perform many tasks (e.g., reply to a message, save an attachment from a message etc.) on a message. Since the messages and the domains were unknown to the users (they had only 1 hour of practice), some of the users had to make multiple attempts to perform the same task. On average, we recorded higher number of user activities on a message in the user study than on a message in the case studies.

Even with all these variabilities in the data sets and the learned parameters, we achieved consistent result with our implicit feedback models. Among all of the models, implicit feedback combined with simple implicit feedback (IFwSIF) always performed the best. For all of the data sets, IFwSIF improves the performance of the TAPE system (as

Table 5.1: Summary of Data sets and the Results from the Experiments.

<b>Data set Attributes</b>	<b>User Study</b>	<b>Sorower-Data</b>	<b>TGD-Data</b>
Total Messages	330	4982	13721
Training Messages	66	1595	9982
Test Messages	270	2391	2741
Avg. Messages per day	NA	17	66
Total Tags	6	43	214
Label Cardinality	1.24	1.23	1.07
<b>Model Parameters:</b>			
<b>STrain Model</b>			
PL	0.74	0.95	0.05
NU	0.08	0.95	0.50
<b>Model Parameter:</b>			
<b>IF Models</b>			
IF Threshold	7	4	5
<b>Results: Performance</b>			
Models (worst to best)	NoIF	STrain	STrain
	<IFwoSIF	<NoIF	<NoIF
	<SIF	<IFwoSIF	<IFwoSIF
	<IFwSIF	<IfwSIFLW	<SIF
	<Online	<SIF	<IFwSIF
		<IFwSIF	<Online
		<Online	
<b>Mistakes per Message:</b>			
NoIF	0.66	0.62	0.43
SIF	0.56	0.51	0.35
IFwoSIF	0.63	0.54	0.37
<b>IFwSIF</b>	<b>0.53</b>	<b>0.47</b>	<b>0.33</b>
IFwSIFLW	–	0.52	–
STrain	0.58	0.75	0.93
Online	0.42	0.27	0.23



compared to its default setting the NoIF model) at least by 50%. Table 5.1 also shows the average mistakes per message for each of the models which shows that IFwSIF makes the minimum number of mistakes per message.

It is also interesting that the performance of the simple implicit feedback model (SIF) closely followed the performance of IFwSIF. Recall that SIF is an aggressive approach that trains as soon as the user changes any one tag on the message. It is likely that SIF is able to recover from some of the mistakes by using the TAPE undo mechanism. As expected, the IFwSIF with learned weights on the implicit feedback events (IFwSIFLW) performs better than no implicit feedback (NoIF) model. The weights on the implicit feedback events need to be learned using a larger validation data set, and the IFwSIFLW model needs to be evaluated on more data sets. This is left as a future work for this thesis.

The STrain model relies on the assumption that the high confidence positive and negative predictions of the TAPE system are likely to be correct. This may be true when the classifiers are already trained on a large number of messages and the distribution of the features in the data does not change. However, as described in Section 2.1, in the online email tagging problem, the distribution of features in the data is often non-stationary. This is why in both of the case studies, STrain shows some benefits at the beginning, but in the long run, performs very badly, even much worse than the NoIF model.

## Chapter 6: Tag-based Email Services

The tag-based information organization approach has been very popular among the online community. A tag is usually a simple non-hierarchical keyword that is meaningful to the user with respect to the objects to which the tag is assigned [41]. The user creates, manages, and applies the tags on email messages in the hope that this might help the user find relevant message or objects at some future time. Some current email clients support tagging and the benefit is that the users can sort the messages by tags. This helps the user retrieve the intended messages in some cases. However, sometimes this ends up creating a very long list of messages that the user needs to sort through. The goal of this chapter is to explore how tags could help with information re-finding and other tasks that users need to perform.

Some of the challenges the email users face are as follows.

- find old messages
- find the destination directory for saving an email attachment
- find the intended file to attach to an outgoing message
- find the intended recipient for an outgoing or forwarded message

One of the goals of this thesis is to make tags generally useful for supporting user workflows. While many operating systems provide easy access to recent objects, the key idea in this chapter is to provide access to recent objects associated with each tag. By specializing for each tag, our hypothesis is that these recent objects lists will become much more valuable to the user. To explore this hypothesis, in this chapter we present the results of a simulation study in which we use the data that we collected in our user study to perform a preliminary assessment of the effectiveness of tag-based access to recent documents, folders, and email addresses. We will refer to these improvements as “tag-based services”, since their effectiveness and ease of use is based on assigning tags to email messages.

## 6.1 Email Services through TAPE

As described in Section 3.1, the TAPE UI provides a TaskBar that includes the tags associated with the message. In this chapter, we explore the idea that clicking on a tag could pop up a menu that offers several potentially-useful actions. Here, we list a set of actions that may be useful.

### *Incoming Email Message:*

- View/Create search folder for the tag (already implemented in TAPE)
- Save attachments – show a list of 3-5 folders that are likely destination folders (based on the chosen tag)
- Forward the message with a list of 3-5 recipients (based on the chosen tag)

### *Outgoing (being composed) Email Message:*

- Attach file – show a list of 3-5 most recently used (MRU) files and folders that are likely to be attached in the current email message (based on the chosen tag)
- Add recipient by selecting from a list of 3-5 email recipients (based on the chosen tag)

## 6.2 Click Cost and Simulation of Tag Services

We assess the benefits of tag services through a simulation study using the user study data. Recall that the user study messages reflect the messages a typical graduate student would receive in everyday life. Some of the messages ask the user to perform tasks that include finding old messages to copy information from, forwarding a message to another recipient, saving an attachment, and sending messages with or without attachment. Each of these tasks requires a certain amount of user effort, and we measure this effort by the number of clicks (click cost) required to perform the task.

The click cost without tag services is the total number of clicks required to perform the tasks during the course of the user study where the user didn't have any of the above specified tag services. For each of the participants, we randomly select moments from the user study sessions where the user is looking for an old message or saving an attachment or looking for a file to attach to a message or looking for a recipient for an outgoing

message. We select 10 instances of each of these tasks for each user. Then we count the total number of clicks by the user to find an old message, to save an attachment, to attach a file to a message and to find the recipient for an outgoing or forwarding message. Finally, we take the average over all 14 participants.

To compute the click cost with tag services, we simulate the user performing the exact same tasks but under an (unimplemented) TAPE user interface that provides these tag actions. Define the following simulation parameters:

- $\eta$  = probability of a predicted tag being correct
- $\gamma$  = probability of a target element being in the MRU list
- $C_m^o$  = click cost of finding a message without tag services
- $C_d^o$  = click cost of finding a directory without tag services
- $C_f^o$  = click cost of finding a file without tag services
- $C_r^o$  = click cost of finding a recipient without tag services
- $C_m^w$  = click cost of finding a message with tag services
- $C_d^w$  = click cost of finding a directory with tag services
- $C_f^w$  = click cost of finding a file with tag services
- $C_r^w$  = click cost of finding a recipient with tag services
- $n_m$  = total number of ‘finding message’ events during the user study
- $n_d$  = total number of ‘finding directory’ events during the user study
- $n_f$  = total number of ‘finding file’ events during the user study
- $n_r$  = total number of finding ‘recipient’ events during the user study
- $C_c$  = click cost of correcting a tag

We can now define the expected total click cost of each type of event.

When searching for an email message, there are four cases to consider. The current message and the target message could both have the correct tag, both have incorrect tags, or one could have the correct tag while the other does not. The expected total click cost of finding messages defined below considers all four cases.

$$E[TC_m^w] = \{\eta \times \eta \times C_m^w + \eta \times (1 - \eta) \times (C_c + C_m^o) + (1 - \eta) \times \eta \times (C_c + C_m^w) + (1 - \eta) \times (1 - \eta) \times (C_c + 1 + C_m^o)\} \times n_m \quad (6.1)$$

Similarly, to compute the expected click cost of finding a directory to save an attachment ( $E[C_d]$ ), finding a file to attach in an email ( $E[C_f]$ ), and finding a recipient for an outgoing or forwarding email ( $E[C_r]$ ), we consider the cases where the current message has the correct or incorrect tag, and also whether the intended item is in the MRU list or not.

$$E[TC_d^w] = \{\eta \times \gamma \times C_d^w + \eta \times (1 - \gamma) \times (C_c + C_d^o) + (1 - \eta) \times \gamma \times (C_c + C_d^w) + (1 - \eta) \times (1 - \gamma) \times (C_c + 1 + C_d^o)\} \times n_d \quad (6.2)$$

$$E[TC_f^w] = \{\eta \times \gamma \times C_f^w + \eta \times (1 - \gamma) \times (C_c + C_f^o) + (1 - \eta) \times \gamma \times (C_c + C_f^w) + (1 - \eta) \times (1 - \gamma) \times (C_c + 1 + C_f^o)\} \times n_f \quad (6.3)$$

$$E[TC_r^w] = \{\eta \times \gamma \times C_r^w + \eta \times (1 - \gamma) \times (C_c + C_r^o) + (1 - \eta) \times \gamma \times (C_c + C_r^w) + (1 - \eta) \times (1 - \gamma) \times (C_c + 1 + C_r^o)\} \times n_r \quad (6.4)$$

Finally, the expected total click cost with tag services

$$E[TC^w] = E[TC_m^w] + E[TC_d^w] + E[TC_f^w] + E[TC_r^w]. \quad (6.5)$$

### 6.3 Simulation Results

During the user study, none of the tag services was implemented. Therefore, we estimate the number of required clicks without tag services as follows.

click cost of finding a message without tag services,  $C_m^o = 5.8$

click cost of finding a directory without tag services,  $C_d^o = 7.5$

click cost of finding a file without tag services,  $C_f^o = 6$

click cost of finding a recipient without tag services,  $C_r^o = 6$

During the user study, the users made tag corrections. To estimate the click cost of correcting a tag, we need to consider three cases. First, the users were able to delete a mispredicted tag with just one click by clicking on the cross button where the TAPE UI shows the tag. Second, the users were able to add a missing tag with two clicks by using the drop down menu that lists all the user-defined tags. Third, in the case of replacing a mispredicted by a missing tag, the users were able to do so with just three clicks. We take average of all these events during the user study and take average over all the participants to estimate the click cost of correcting a tag.

click cost of correcting a tag,  $C_c = 2$

(6.6)

With the tag services described above, the user should be able to perform the tasks with fewer clicks. For example, if the intended directory, file, or recipient is indeed in the (MRU) list for the corresponding tag, then the user should be able to save attachment, attach a file or add a recipient in just 2 clicks (e.g., click on the tag and then click on the item in the (MRU) list). Similarly, for finding an old message, the user could click on the tag to view/create search folder for that tag and then click on the intended message<sup>1</sup>.

---

<sup>1</sup>For simplicity, we ignore the cost of visual search and scrolling a menu in this computation. Search and scrolling will add more cost for performing tasks without tag services than with tag services.

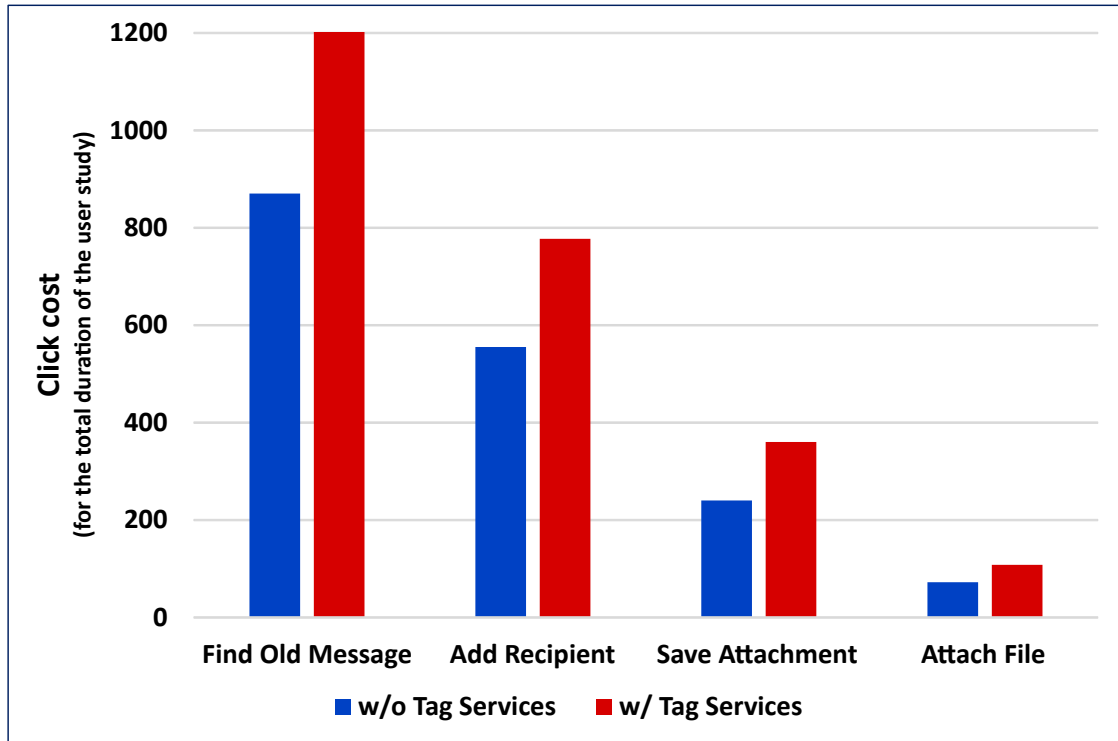


Figure 6.1: Expected click cost without and with tag services computed for the 270 message in the user-study for  $\eta = 0.0$  and  $\gamma = 0.0$ .

click cost of finding a message with tag services,  $C_m^w = 2$

click cost of finding a directory with tag services,  $C_d^w = 2$

click cost of finding a file with tag services,  $C_f^w = 2$

click cost of finding a recipient with tag services,  $C_r^w = 2$

In order to compare the costs with and without tag services, we need to know the values of the  $\gamma$  and  $\eta$  parameters. Because these varied from user to user and depend on the exact order in which the users perform actions, these are difficult to compute on the actual user study data. For purposes of this initial assessment, we decided to simply plug in a set of fixed values for these parameters in our simulation.

Figure 6.1–6.4 compare the expected total cost without and with tag services for the 270 messages in the user study for different values of  $\eta$  and  $\gamma$ . Note that  $\eta = 0.0$  implies that the tag on the message is never correct, and  $\gamma = 0.0$  implies that the intended item will never be on the (MRU) list of tag specific actions. This is an adversarial (e.g., worse than random) case where tag prediction is not useful at all, and hence, the total click cost with tag services is worse than the total cost without tag services (Figure 6.1). The benefits of tag services start to appear as the tag predictions and the MRU lists becomes 50% accurate. However, as described above, a more realistic case is that the tag is correct about 80% of the time (which we have seen in our regular usage of TAPE). Figure 6.3 shows some clear gain on click cost in the presence of tag services. In some cases, the click cost is reduced by about 50%, computed over the 270 messages in the user study. Obviously the maximum benefit of tag services is attained when  $\eta = 1.0$  and  $\gamma = 1.0$  (Figure 6.4).



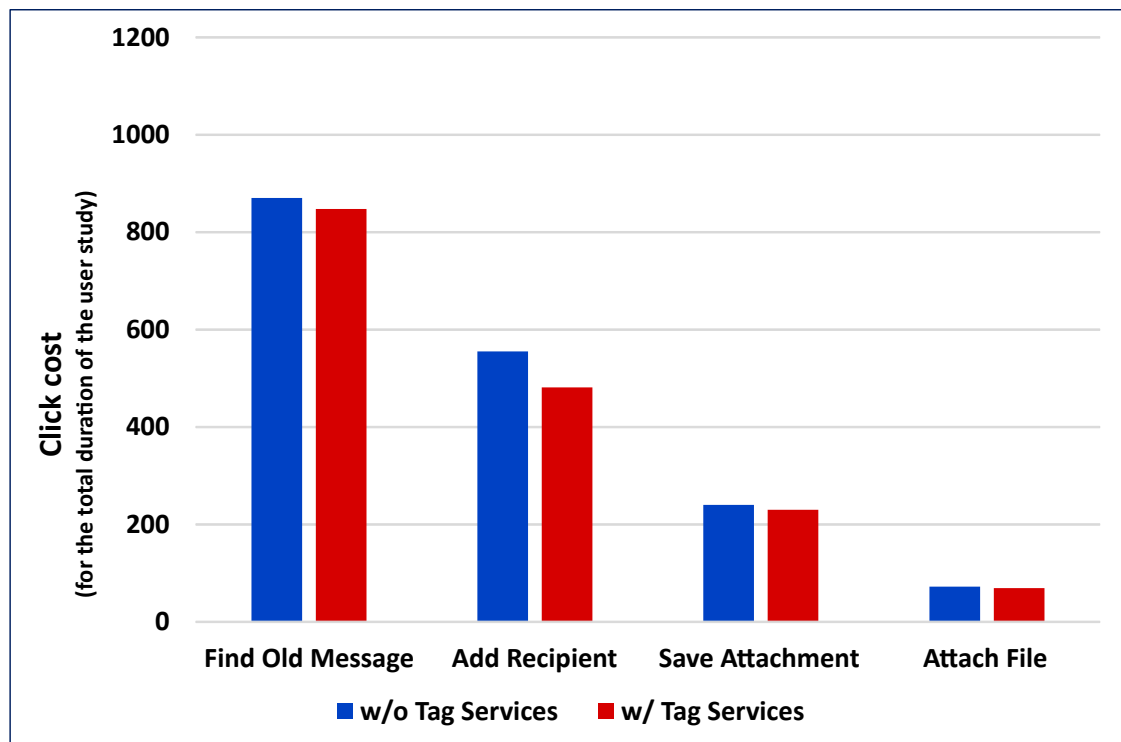


Figure 6.2: Expected click cost without and with tag services computed for the 270 message in the user-study for  $\eta = 0.5$  and  $\gamma = 0.5$ .

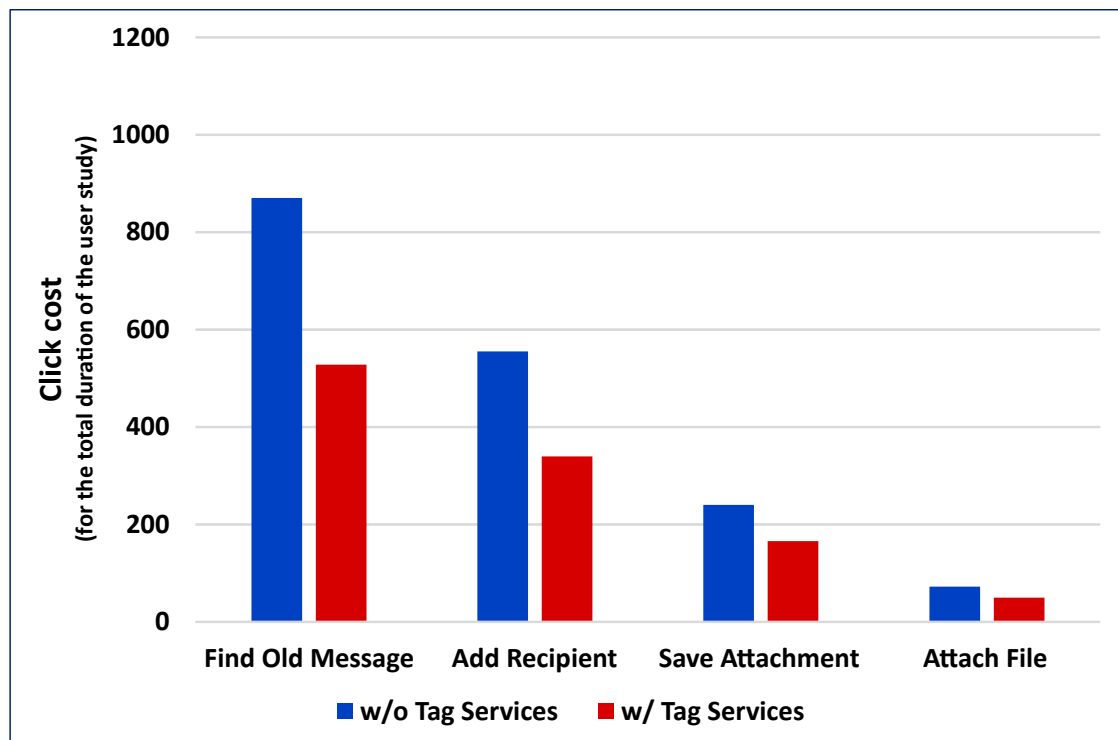


Figure 6.3: Expected click cost without and with tag services computed for the 270 message in the user-study for  $\eta = 0.8$  and  $\gamma = 0.7$ .

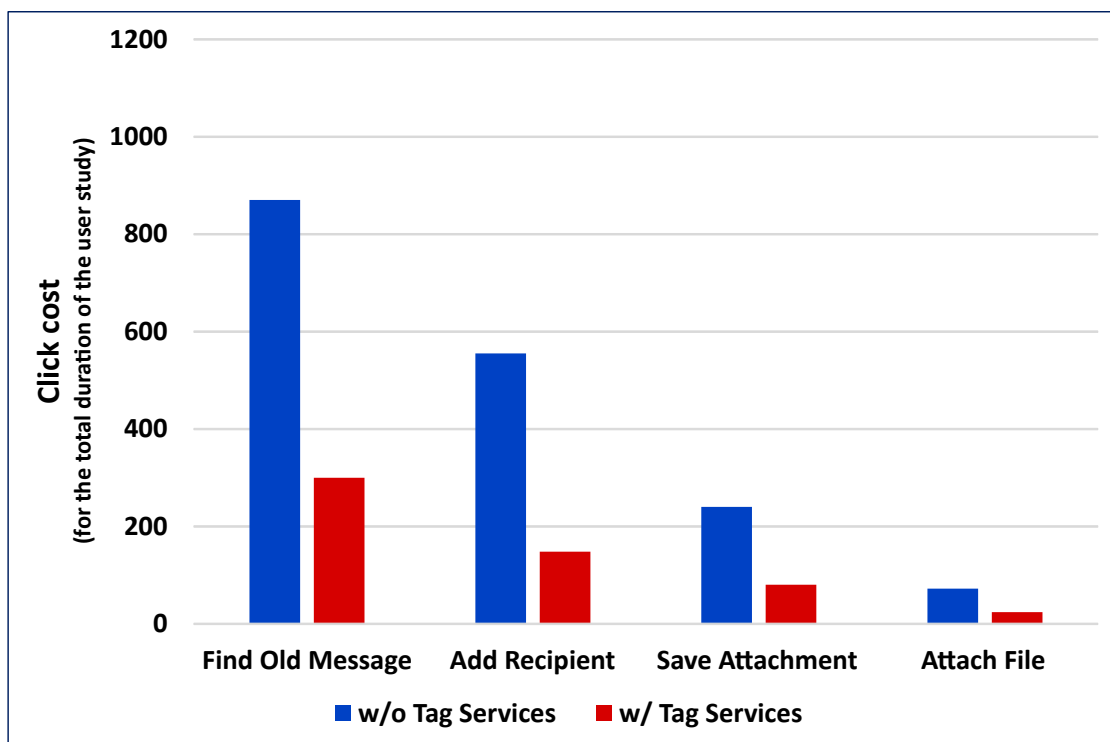


Figure 6.4: Expected click cost without and with tag services computed for the 270 message in the user-study for  $\eta = 1.0$  and  $\gamma = 1.0$ .

## 6.4 Summary

The preliminary study shows that tag services have potential to greatly reduce the number of clicks required to perform these four common tasks: finding a relevant email message, deciding where to save an attachment, finding a file to attach, and choosing a recipient for an outgoing email message. This study provides important evidence that tag-based services should be implemented for Outlook and for other desktop applications. This is an important area for future research.

## Chapter 7: Conclusion and Future Work

Tagging is an important tactic for information organization. Automatic tagging of email messages provides a personalized solution for organizing user mailboxes. Any tagging system must cope with the problem of incomplete feedback, both because users will forget to provide feedback and because users will generally only provide corrective feedback rather than confirming that predicted tags are correct. The results of our investigation through a lab-controlled user study and the case studies with two knowledge workers make it clear that an email tagging system can benefit from even a simple implicit feedback confirmation system. We studied two sources of implicit feedback. Simple Implicit Feedback (SIF) captures the case where the user corrects one tag on an email message. The remaining tags (absent or present) are thereby confirmed as being correct. Implicit Feedback without SIF (IFwoSIF) is based on the observation that the longer a user interacts with an email message, the more likely they are to notice and correct any incorrect tags. Both of these sources of feedback were shown to provide substantial additional training examples to the classifier, which produced good reductions in prediction errors. Their combination, IFwSIF, provided even better error reductions. The error reductions were obtained despite the fact that the additional training examples produced by these implicit feedback mechanisms had a fairly high rate of erroneous tags. We therefore recommend that machine-learning-based tagging systems should incorporate implicit feedback mechanisms to extract more information from the user's interaction with the user interface.

With the help of a simulated study (Chapter 6), we also showed that tags can contextualize the tagged objects. In the case of a tag-based email system, the presence of some tag(s) on a message could indicate the user-work context (e.g., project/task), and, therefore, the system can proactively suggest relevant resources (e.g., relevant messages or files) for the current context. Our preliminary study shows that tag services have potential to greatly reduce the number of clicks required to perform several common tasks on an email message.

## 7.1 Future Work

The results of the experiments with the user study and the case studies are very encouraging. There is also additional room to improve the effectiveness of implicit feedback. First, our current approach treats all forms of implicit feedback events as being equally informative. Although in IFwSIFLW model, we learned the weights on the implicit feedback events, with a larger sample, and perhaps using eye-tracking, we could determine which IF events are more likely to cause the user to notice incorrect tags. Second, in real applications of TAPE, some tags are much more common than others and so have many more positive training examples. In addition, new tags are introduced frequently, so that some classifiers have very few training examples. Of course the accuracy of a classifier increases as it is given more examples. Therefore, one might expect that the implicit feedback threshold might need to be higher for messages with poorly-trained tags and lower for well-trained tags. It would be interesting to explore this hypothesis. Third, our study shows that the training examples created by IFwSIF still contain many incorrect tags. This suggests that we should either modify the confidence weighted classifier to make less aggressive updates when trained on these examples or else switch to an online learning algorithm that is more robust to label noise, such as AROW [9].

There are also potential improvements in the user interface that could encourage the user to provide more explicit feedback. For example, because feedback is most useful on tags for which the classifier has low confidence, we could provide a visual cue to the user (e.g., by changing the color of the tag button) to call attention to those low-confidence tags. We could add a drop-down option for the user to confirm that the tag is correct. If the user selected this, then the tag color could change back to a neutral or even a positive color.

Finally, as we have shown in our simulated study in Chapter 6, tags could support functions beyond simple email organization. In the original TaskTracer system, when the user saved an attachment (or opened a file to attach to an email message), the “current task” of the user was consulted to suggest appropriate task-related folders in the open/save dialogue box. A similar idea could be implemented using tags. To save an attachment, the user could click on a tag to reveal a “Save Attachment” drop-down menu item. The folders associated with that tag could be presented in the open/save dialogue. A similar interaction would make adding attachments to outgoing messages easier. The

more functions that tags support, the more motivation the user has to fix incorrect tags, and the more the cost of tagging is repaid in improved productivity. This suggests that we should implement the tag services in TAPE and conduct a user study or case studies to evaluate and quantify the benefits of tag services.

## Bibliography

- [1] A. Agrawala. Learning with a probabilistic teacher. *Information Theory, IEEE Transactions on*, 16(4):373–379, Jul 1970.
- [2] Joshua Akehurst, Irena Koprinska, Kalina Yacef, Luiz Pizzato, Judy Kay, and Tomasz Rej. Explicit and implicit user preferences in online dating. In *Proceedings of the 15th international conference on New Frontiers in Applied Data Mining, PAKDD'11*, pages 15–27, Berlin, Heidelberg, 2012. Springer-Verlag.
- [3] Morgan Ames and et al. Why we tag: motivations for annotation in mobile and online media. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems*, pages 971–980. ACM Press, 2007.
- [4] R. Bekkerman, A. McCallum, and G. Huang. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. *Center for Intelligent Information Retrieval, Technical Report IR*, 418, 2004.
- [5] Paul-Alexandru Chirita, Stefania Costache, Julien Gaugaz, and Wolfgang Nejdl. Desktop context detection using implicit feedback. *Personal Information Management: Now That We’re Talking, What Are We Learning?*, page 24, 2006.
- [6] Mark Claypool, David Brown, Phong Le, and Makoto Waseda. Inferring user interest. *IEEE Internet Computing*, 5:32–39, 2001.
- [7] W. W. Cohen. Learning Rules that classify E-mail. In *AAAI Spring Symposium in Information Access*, 1996.
- [8] Koby Crammer and Claudio Gentile. Multiclass classification with bandit feedback using adaptive regularization. *Machine Learning*, 90(3):347–383, 2013.
- [9] Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In *Advances in Neural Information Processing Systems 22*, pages 414–422, 2009.
- [10] Elisabeth Crawford, Judy Kay, and Eric McCreath. IEMS - the intelligent email sorter. In *Proceedings of the Nineteenth International Conference on Machine Learning, ICML '02*, pages 83–90, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.




- [11] Christoffer Davidsson and Simon Moritz. Utilizing implicit feedback and context to recommend mobile applications from first use. In *Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation*, CaRR '11, pages 19–22, New York, NY, USA, 2011. ACM.
- [12] Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew Mclaughlin, Lida Li, and Jonathan L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *In IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82. ACM Press, 2005.
- [13] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *International Conference on Machine Learning (ICML)*, 2008.
- [14] Nicolas Ducheneaut and Victoria Bellotti. E-mail as habitat: an exploration of embedded personal information management. *Interactions*, 8(5):30–38, September 2001.
- [15] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, April 2005.
- [16] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. September 2003.
- [17] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.
- [18] Jacek Gwizdka and Mark H. Chignell. Individual differences and task-based user interface evaluation: a case study of pending tasks in email. *Interacting with Computers*, 16:769–797, 2004.
- [19] Matthias Hutterer. Enhancing a job recommender with implicit user feedback. In *Masters Thesis*, 2011.
- [20] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA, 2010. ACM.
- [21] Victoria Keiser and Thomas G. Dietterich. Evaluating online text classification algorithms for email prediction in tasktracer. In *Conference on Email and Anti-Spam*, 2009.

- [22] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON '01*, pages 8–. IBM Press, 2001.
- [23] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, John Riedl, and High Volume. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40:77–87, 1997.
- [24] Irena Koprinska, Josiah Poon, James Clark, and Jason Chan. Learning to classify e-mail. *Inf. Sci.*, 177(10):2167–2187, May 2007.
- [25] Alfred Krzywicki and Wayne Wobcke. Incremental e-mail classification and rule suggestion using simple term statistics. In *Proceedings of the 22nd Australasian Joint Conference on Advances in Artificial Intelligence, AI '09*, pages 250–259, Berlin, Heidelberg, 2009. Springer-Verlag.
- [26] Danielle H. Lee and Peter Brusilovsky. Reinforcing recommendation using implicit negative feedback. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH, UMAP '09*, pages 422–427, Berlin, Heidelberg, 2009. Springer-Verlag.
- [27] Wendy E. Mackay. More than just a communication system: diversity in the use of electronic mail. In *Proceedings of the 1988 ACM conference on Computer-supported cooperative work, CSCW '88*, pages 344–353, New York, NY, USA, 1988. ACM.
- [28] Masahiro Morita and Yoichi Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '94*, pages 272–281, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
- [29] Denis Parra-Santander and Xavier Amatriain. Walk the talk: Analyzing the relation between implicit and explicit feedback for preference elicitation. In *User Modeling, Adaptation & Personalization*. Springer, July 2011.
- [30] L. Pizzato and University of Sydney. School of Information Technologies. *Learning User Preferences in Online Dating*. Technical report (University of Sydney. School of Information Technologies). University of Sydney, School of Information Technologies, 2010.
- [31] Filip Radlinski and Thorsten Joachims. Evaluating the Robustness of Learning from Implicit Feedback. In *ICML workshop on Learning in Web Search*, 2005.

- [32] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Knowledge Discovery and Data Mining*, pages 239–248, 2005.
- [33] Jason D. M. Rennie. ifile: An application of machine learning to e-mail filtering. In *Proc. KDD Workshop on Text Mining*, 2000.
- [34] H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *Information Theory, IEEE Transactions on*, 11(3):363–371, Jul 1965.
- [35] Richard Segal and Jeffrey O. Kephart. Incremental learning in SwiftFile. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML 2000, pages 863–870, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [36] Michael Slater, Thomas Dietterich, and Mohammad Sorower. Tape: An integrated machine learning system for email tagging, in preparation.
- [37] Frank Curtis Stevens. *Knowledge-based Assistance for Accessing Large, Poorly Structured Information Spaces*. PhD thesis, Boulder, CO, USA, 1993. UMI Order No. GAX93-20482.
- [38] Stefanos Vrochidis, Ioannis Kompatsiaris, and Ioannis Patras. Utilizing Implicit User Feedback to Improve Interactive Video Retrieval. *Advances in Multimedia*, 2011:1–18, January 2011.
- [39] Steve Whittaker, Victoria Bellotti, and Paul Moody. Revisiting and Reinventing Email. *HCI Special Issue on Email*, 20(1):1–9, June 2005.
- [40] Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *CHI 96 Conference On Human Factors In Computing Systems*, pages 276–283. ACM Press, 1996.
- [41] Wikipedia. Tag (metadata) — wikipedia, the free encyclopedia, 2013. [Online; accessed 12-December-2013].
- [42] Lei Zhang, Xiang-Wu Meng, Jun-Liang Chen, Si-Cheng Xiong, and Kun Duan. Alleviating cold-start problem by using implicit feedback. In *Proceedings of the 5th International Conference on Advanced Data Mining and Applications*, ADMA '09, pages 763–771, Berlin, Heidelberg, 2009. Springer-Verlag.

## APPENDICES

## Appendix A: Subject Recruitment Flyer for the User Study



**Inbox Overload 2 !?!**

We have developed new software for helping you manage your email. And we are seeking volunteers to participate in a study to measure how well it works.

Earn \$60 participating in a study of our newest email management tools.

If you'd like to be part of our study, or learn more about our research, please contact us:

[inbox2@eecs.oregonstate.edu](mailto:inbox2@eecs.oregonstate.edu)  
<http://research.engr.oregonstate.edu/inbox2>

Participation requirements: be an adult email user, receive 20+ emails per day, regularly use categories, tags, labels, or folders to organize your email.

The principal investigator and conductor of the EP2 Inbox Overload 2 study is Prof. Thomas Dietterich, Intelligent Systems Research Group, EECS, Oregon State University. If you have any questions, please contact the Research Coordinator, Michael Slater, via email at [slater@eecs.oregonstate.edu](mailto:slater@eecs.oregonstate.edu) or phone at 541-737-5726. You can also contact Dr. Dietterich via email at [tgd@eecs.oregonstate.edu](mailto:tgd@eecs.oregonstate.edu) or via phone at 541-737-5559.

Figure A.1: A snapshot of the flyer used to recruit subjects for the user study.

## Appendix B: Subject Eligibility Questionnaire for the User Study

Question 1: Do you meet the eligibility criteria for this study?

- Adult email user (18 years of age or older)
- Receive 20 or more email messages per day
- Regularly use tags, categories, labels, or folders to organize your email

Yes       No

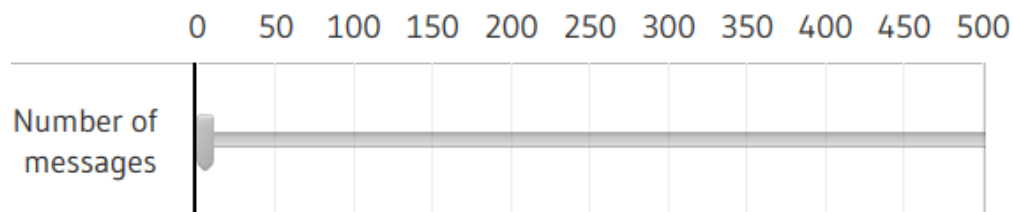
Question 2: What email software do you use as your main email client?

- Microsoft Outlook       Thunderbird       Mac OS X Mail.app  
 Gmail or Google Apps       Yahoo Mail       Other (please fill in below):

Question 3: Which of the following do you regularly use to organize your email (please check all that apply)?

- Tags       Categories       Labels       Folders  
 Flags       Stars       None of the above

Question 4: In the past two weekdays, approximately how many emails did you receive?



Question 5: Out of the messages you received in the past two weekdays, how frequently

did you add a tag, category or label? If email tags, categories, or labels are added automatically by a rule or other means, please include the count of those messages in your estimate.

- I don't know what you mean by tag, category or label
- Never       Sometimes       Often

Question 6: Out of the messages you received in the past two weekdays, how frequently did you move a message to an email folder? If email was automatically moved to a folder by a rule or other means, please include the count of those messages in your estimate.

- Never       Sometimes       Often

Question 7: In the past two weekdays, how many different tags, categories, or labels did you use?

- None       Less than 10       More than 10

Question 8: In the past two weekdays, how many different email folders did you use (find messages in, move message to, etc.)?

- None       Less than 10       More than 10

Question 9: Can you please describe how you use tags, labels, categories, and/or folders to organize or manage your email?

## Appendix C: Sample Email Messages from the User Study

### Sample Message: 1

Sender: Events-Gardening <events@humblekitchen.net>  
Subject: organic gardening workshop! December 28-29

Hello gardeners ,  
Thank you for registering for the Organic Gardening workshop  
2014. Here is the workshop details .

When: December 28-29, 2014  
Where: TI Convention Center

We will send you the detail schedule and agenda a week prior to  
the workshop .

We will need some volunteer to help us with the 'Organic  
Gardening' t-shirt design . I have downloaded some templates from  
the web and have attached them herewith .

Can someone please make a design for us? We need to send it to  
the press by next week! For your kind work , you will receive 10  
different seedlings of your choice !

Thanks for your time , and see you all soon .  
—Dan



**Sample Message: 2**

Sender: Dixon <d@techfuture.us>

Subject: practice test

Hi class,

I have attached a practice test for algebra that you may want to study for the upcoming midterm exam.

The midterm will be similar except that we have covered additional materials such as set theory, linear programming and part of number theory.

Good luck,

—D

Attachment: Algebra II Practice Test.pdf

**Sample Message: 3**

Sender: Bob <bob@appqualify.com>

Subject: FW: Eat Healthy Chart!

Hi,

I was looking for tools and resources that may help you better understand nutrition and the important role healthy eating plays in maintaining a healthy weight.

I have attached a chart that you will find useful. Use this chart to track what your family is doing to eat healthy and move more each week.

SAVE THIS PDF. This will help you (and your doctor) to decide how to proceed with your treatments.

get well soon!

—Don

Attachment: tip-eat-healthy-chart.pdf

**Sample Message: 4**

Sender: Dawn <dawn@waldenorganic.org>

Subject: Econ 103 project proposal

Hi,

I look forward to working with you for our economics class project. As we discussed this afternoon, I have written an initial draft of the proposal to evaluate the fiscal impact of super bowl on local business, and the growth of the city as a whole.

We should focus on Economic impact analysis examines the regional implications of an activity in terms of three basic measures: sales or output, earnings, and job creation.

Could you please review the attached draft and WRITE DOWN your name and SEND the document back to me?

cheers,

—Dawn

**Sample Message: 5**

Sender: Becky <becky@appqualify.org>

Subject: Family Movie Night – First Friday Family Film

Hello friends,

We have organized a movie night this Friday. Since this is our first attempt, we have only invited a few selected people, friends and family we know. You are cordially invited (with your family). Here is the details:

Event Type: Family/All ages

Date: 01/03/2015

Start Time: 7:00 PM

End Time: 8:30 PM

Library: Easttown Library & Information Center

Location: Arronson & Kohn Rooms

Description: All ages. Doors open at 6:45 p.m.

Other: Join us the first Friday evening of each month for a family-friendly movie. Movies will range from animated features to retro classics. Movies are free, so gather your family and friends, bundle the kids in their pajamas, and come watch a movie on our big screen.

Email this event to a friend, so they can register for it as well.

Please feel free to bring any snacks if you would like!

Please RSVP by sending a reply to this email. Even if you cannot attend, please let us know in your **REPLY**.

Looking forward to see you there!

—Becky

**Sample Message: 6**

Sender: Joshua <joshua@bountybank.com>

Subject: plants to grow indoor

hi,

I have finally started indoor gardening – thanks to you :)

Can you please FORWARD me the list of plants you have that can be grown indoor in containers? I have started with mint and I love it!

Please also PRINT that email so we can distribute that among the gardeners... I'm sure they will like it.

cheers

Joshua

**Sample Message: 7**

Sender: Sam <sam@forestandbark.net>

Subject: opportunity cost definition

Hi,

I have been looking around for a good definition and example for 'opportunity cost'. You said you have a web link somewhere in your email. Can you please COPY/PASTE that link in REPLY or just FORWARD that email to me?

see you

**Sample Message: 8**

Sender: Oliver <oliver@familymates.com>  
Subject: calculus honors seminar that the professor forwarded?

Hello ,  
Could you please FORWARD me the email the professor sent about the seminar? I think that was about calculus or math or something . . . .

oh, Sam said he needs that too. Could you please also FORWARD that email to him (sam@forestandbark.net)?

Oliver

**Sample Message: 9**

Sender: Dawn <dawn@waldenorganic.org>  
Subject: trigonometry homework? attachment missing?

Have you started working on the trigonometry homework? I think I solved the bonus problem. I have  $\sec \theta = 1$  and  $\cot \theta = 2$ . Does this match your results?

BTW, I think the professor forgot to attach the homework questions pdf in that email. Would you please CHECK the email you received and see if you received any attachment.

If you have the attachment, please FORWARD that email to me. If you also do not have the attachment with that email, please LET ME KNOW. I'll then send an email to the professor.

have a nice weekend!  
Dawn

**Sample Message: 10**

Sender: Robin <robin@humblediet.org>

Subject: meeting request

Dear student representative:

Linda Jones, Director of Career Services at The OSU Moritz College of Law suggested I contact you given my interest in exploring a career advocating for students who are lagging behind. I would appreciate the opportunity to spend 30 minutes with you for an informational interview to help guide my curricular and career decisions.

Can we meet on Friday at 11am? Please let me know by REPLYING to this email.

Thank you for your consideration. You have done what I hope to do: graduate from the Moritz College and develop a successful and rewarding education practice. Any guidance you may provide would be most appreciated.

I have also attached the agenda from our last meeting. SAVE THE ATTACHMENT, in case if you want to review this.

Sincerely,  
Robin

