EmFore: Online Learning of Email Folder Classification Rules

Mukul Singh  
Microsoft  
Delhi, India  
singhmukul@microsoft.com

José Cambronero  
Microsoft  
Redmond, USA  
jcambronero@microsoft.com

Sumit Gulwani  
Microsoft  
Redmond, USA  
sumitg@microsoft.com

Vu Le  
Microsoft  
Redmond, USA  
levu@microsoft.com

Gust Verbruggen  
Microsoft  
Keerbergen, Belgium  
gverbruggen@microsoft.com

ABSTRACT
Modern email clients support predicate-based folder assignment rules that can automatically organize emails. Unfortunately, users still need to write these rules manually. Prior machine learning approaches have framed automatically assigning email to folders as a classification task and do not produce symbolic rules. Prior inductive logic programming (ILP) approaches, which generate symbolic rules, fail to learn efficiently in the online environment needed for email management. To close this gap, we present EmFore, an online system that learns symbolic rules for email classification from observations. Our key insights to do this successfully are: (1) learning rules over a folder abstraction that supports quickly determining candidate predicates to add or replace terms in a rule, (2) ensuring that rules remain consistent with historical assignments, (3) ranking rule updates based on existing predicate and folder name similarity, and (4) building a rule suppression model to avoid surfacing low-confidence folder predictions while keeping the rule for future use. We evaluate on two popular public email corpora and compare to 13 baselines, including state-of-the-art ILP and transformer-based approaches. We find that EmFore performs significantly better, updates four orders of magnitude faster, and is more robust than existing methods and baselines.

CCS CONCEPTS
• Computing methodologies → Online learning settings; Rule learning; • Information systems → Email.

KEYWORDS
Email Classification, Online Learning, Learning by Examples

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom  
© 2023 Copyright held by the owner/authors(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-9424-5/23/10  ... $15.00  
https://doi.org/10.1145/3583780.3614863

1 INTRODUCTION
Email remains one of the most important forms of digital communication. Professional users receive over 100 emails per day on average [1]. With storage becoming cheaper, these emails are rarely deleted [9]. Managing both unread and read emails in an inbox thus becomes a time-consuming task. Furthermore, spending more time on email has been found to correlate with lower perceived productivity and higher measures of stress in professionals [30].

Most email services provide tools—backed by research—that help users manage their inbox. Spam prediction reduces inbox clutter. Estimating email importance [1] helps users focus on important emails in the Focused inbox in Outlook and Priority inbox in Gmail. Search helps users to quickly find specific emails [28].

To enable more efficient folder management, modern email clients allow users to create rules for moving emails into folders based on simple properties, for example, the subject containing a specific phrase. These rules are a powerful tool for email management, but authoring them manually can be difficult for novices and tedious for advanced users.

Automatically categorizing emails in folders has attracted attention in research [10, 14, 15, 17]. Early systems were based on learning over a training set and freezing the models for inference [7, 18, 31]. More recent systems [10, 17] allow for updating rules but these are over large batches and not real time. Many of these same approaches frame categorization as classification and fail to generate a symbolic rule, which users can inspect and integrate into their email client. On the other hand, inductive logic programming based approaches [14, 15] do support symbolic rule generation, but they have not supported online learning (i.e., updating rules after every incoming email) as required in the email management setting. We hypothesize that these limitations have contributed to such functionality being absent in email clients.

In this paper, we introduce the first online system to learn rules for email folder classification by demonstration. Our system, called EmFore, observes a user moving emails into folders and learns a rule for each folder. EmFore can update rules in real time with each new incoming email, allowing incremental improvements in performance. This approach draws inspiration from the successful application of the programming by example paradigm in commercial products like Excel [23] and Visual Studio [32].
Inspired by the rule language in popular email clients, the rules learned by EmFore consist of propositions describing email properties. EmFore uses generalization and specialization to inductively update these rules after each new email. To do this efficiently, we (1) create an abstraction of the state of the inbox, (2) devise an updating algorithm based on greedily ranking candidate predicates, and (3) use rule suppression to reduce the number of false positives.

In addition to generating symbolic rules and supporting online rule updates, EmFore also addresses the following gaps highlighted in a recent survey on automated email classification [34]:

1. **Dynamic updating of the feature space.** We do not use a fixed feature representation of emails. New propositions are generated for each incoming email and the feature space is thus automatically extended, when necessary.
2. **Reducing the false positive rate.** We show that rule length is an effective proxy for folders not having an intended rule. EmFore can use this information to abstain from suggesting predictions made by these rules.
3. **Concept drift.** Users can change the scope of folders over time, which leads to concept drift. As soon as a wrong classification is detected by the user, EmFore instantly updates the associated rule to be consistent with all mails in the folder. We thus address cases of concept drift where the scope becomes more general.
4. **Deep learning.** We introduce four transformer-based approaches as neural baselines, an approach missing in the email classification task literature, and show that our symbolic rule learner outperforms these baselines.

In summary, we make the following contributions:

- We present a novel online algorithm for learning email folder classification rules from a few email examples.
- We perform an extensive evaluation of EmFore on two datasets, both in online and offline settings. We find that it outperforms the next best system (Alecsa) by up to 9 points in correct decision rate in the online setting, while learning up to 3 orders of magnitude faster. We release EmFore-labeled results for future use at https://github.com/microsoft/prose-benchmarks/tree/main/Emfore.
- In addition to comparing to existing approaches, we address an existing gap in the literature and implement four neural approaches based on related tasks. EmFore outperforms these approaches by 10 - 15 points in correct decision rate, while learning up to 4 orders of magnitude faster.
- We show that we can configure rule suppression and mail retention in EmFore to reduce false positive rates and adapt to concept drift. In addition, EmFore is adaptable to different email clients (based on expressiveness of rules).

## 2 PROBLEM STATEMENT

We consider an online setting in which an ordered stream of mail and their associated folder \((m_1, f_1), \ldots, (m_t, f_t)\) are given and the model has to predict the folder \(f_{t+1}\) associated with mail \(m_{t+1}\). If this prediction is incorrect—as indicated by the user—the model is allowed to relearn. Any method can be evaluated in this setting by learning the model from scratch after incorrect predictions.

In order to support integration in popular email clients, we consider the model to consist of rules that are supported by such clients. A rule is a formula in propositional logic where propositions are interpreted with respect to emails. If a rule \(R\) evaluates to true for an email \(m\), we say that the email satisfies the rule and write \(m \models R\).

**Example 1.** The rule \(\text{InFrom}(\text{"strawbale"}) \lor \text{InTo}(\text{"strawbale"})\) consists of two propositions and is satisfied by emails with "straw" in at least one of the sender or receiver fields.

By imposing an order on different rules for different folders, each email is placed into exactly one folder. Let \(\mathcal{R} = \{(R_i, f_i)\}\) be an ordered list of rule–folder pairs with \(R_i\) denoting a rule for folder \(f_i\). A mail \(m\) is assigned to the first folder \(f_i\) such that \(m \models R_i\) and we write \(\mathcal{R}(m) = f_i\). If no rule holds for a mail, it defaults to the special inbox folder. The last element of \(\mathcal{R}\) is thus always (true, inbox). We say that \(\mathcal{R}\) is consistent with emails \(\{(m_i, f_i)\}\) if \(\mathcal{R}(m_i) = f_i\) for all \(i\). Rules are thus an extension of decision lists where the assigned value is not restricted to booleans [38].

## 3 APPROACH

Our system takes inspiration from mathematical induction. Let \(\mathcal{R}\) be a set of rules consistent with the current emails. If the prediction \(\mathcal{R}(m_*) = f\) for a new email \(m_*\) is wrong, as indicated by the user moving the email to folder \(f^*\), we update the rule to be consistent with all previous emails and the new email. We introduce three components for doing so: a state \(S\) that tracks candidate propositions for each folder, a space of rules over which \(\mathcal{R}\) is learned, and an algorithm for updating \(\mathcal{R}\). We will describe each component in detail in the following subsections.

### 3.1 State

The state keeps track of the candidate propositions \(S_f\) for each folder \(f\) and ensures that every proposition \(p \in S_f\) is satisfied by an email \(m_i\) in \(\{(m_i, f_i)\}\) if and only if \(f_i = f\). A proposition consists of a logical predicate that can be evaluated on a mail and form the building blocks of our rules. Not every proposition must satisfy all mails in the folder. Any folder with an empty set of candidate propositions cannot be covered by a rule. All propositions that are constructed but not part of the state (because they are satisfied by emails in multiple folders) are kept in a stateful variable called \(P_{all}\).

Candidate propositions are generated for an email from a set of templates by substituting a placeholder \(e\) with a string constant. Table 1 shows a list of supported templates and how they generate propositions. These templates were selected as the union of those used in two

<table>
<thead>
<tr>
<th>Template</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{To}(\text{&quot;<a href="mailto:strawbale@crest.org">strawbale@crest.org</a>&quot;}))</td>
<td>To(&quot;<a href="mailto:absteen@dakotacom.net">absteen@dakotacom.net</a>&quot;)</td>
</tr>
<tr>
<td>(\text{InTo}(\text{&quot;strawbale&quot;}))</td>
<td>InTo(&quot;crest&quot;)</td>
</tr>
<tr>
<td>(\text{InFromOrTo}(\text{&quot;strawbale&quot;}))</td>
<td>InFromOrTo(&quot;dakotacom&quot;)</td>
</tr>
</tbody>
</table>

Candidate propositions for each folder are ranked to allow greedy selection of promising ones when building rules. This ranking takes into account: (1) the similarity between the string constants in the proposition and the folder name, (2) the average similarity to string constants of the current rule for that folder, and (3) the type of...
propagation. Similarities are computed with Jaro-Winkler string similarity. Each of these properties yields a score, which are summed to obtain a final score. The heuristic scores of proposition templates are shown in Table 1. Finally, we compute the final proposition score as a weighted sum of the individual scores. Section 4.5 explains how these weights are learned. The learned weights show that folder name similarity is the most important feature, followed by rule constants similarity. Template score contributes least to the final predicate rankings.

Example 3. Given the rule in Example 1 for a folder “straw” the candidates from Example 2 are ranked as follows (from better to worse):

\[
\begin{align*}
&\text{InTo(“strawbale”) > InFromOrTo(“strawbale”) >} \\
&\text{To(“strawbale@crest.org”) > InTo(“crest”) >} \\
&\text{To(“absteen@dakotacom.net”) > InFromOrTo(“dakotacom”)}
\end{align*}
\]

Whenever an email \((m_i, f_i)\) comes in, propositions \(P_j\) are exhaustively generated, we add these propositions to \(S_{f_i}\) while maintaining the ranking and remove them from \(S_{f_j}\) where \(f_i \neq f_j\). This process happens before our rule updates, described in the next section, which then handles necessary changes to existing rules.

### 3.2 Rule Space

We limit each folder \(f\) to a single rule \(R_f\), which must be in disjunctive normal form (DNF). As every logical formula can be written in DNF, we do not lose expressivity. However, some email clients cannot represent all logical formulas and thus will not be able to represent all rules. Our algorithm can be easily adapted to support other forms, for example, we can disallow negation. We study expressivity in our evaluation (RQ5).

To learn these DNF rules, EmFore relies on generalization and specialization. Let \(c_1 \lor c_2\) be a rule with \(c_1 = p_1 \land p_2\) and \(c_2 = p_3\). Adding disjuncts \((c_i)\) generalizes a rule, as this allows it to match more emails, adding conjuncts \((p_i)\) on the other hand specializes it, as it will match fewer emails. In a rule with the fewest disjuncts consistent with all mails in a folder, there are emails uniquely satisfied by each of the disjuncts. We denote these emails with \(u(c_i)\).

### 3.3 Updating Rules

When a new email \((m_i, f^*)\) comes in, each of the rules \(R_f \in \mathcal{R}\) may be updated. The full algorithm is shown in Figure 2. If \(m_i \not\models R_f\), meaning the email is not covered by the current rule for folder \(f^*\), then it requires generalization (the cover function, lines 12–28). Any rule \(R_f\) with \(f \neq f^*\) and \(m_i \models R_f\), meaning a rule from a different folder \(f\) incorrectly covers the new email, requires specialization (the uncover function, lines 30–51). The ideas around specialization and generalization for learning rules based on examples have been explored in ILP [33]. We extend these ideas to work in an online learning system for the email domain.

![Figure 1: Generalization and specialization steps for a mail that should satisfy folder1 but instead satisfies folder3. First, we try to replace propositions with candidates from the state such that folder1 satisfies the mail (generalize) and folder3 does not (specialize). If no replacements are found, we extend the rule with new disjuncts (generalize) or conjuncts (specialize).](image)

![Figure 2: Online Learning of Email Folder Classification Rules CIKM '23, October 21–25, 2023, Birmingham, United Kingdom](image)

### Table 1: Proposition templates supported by EmFore, inspired by six popular email clients. A generator substitutes \(c\) with a string constant to obtain a proposition. Constants are produced by a tokenizer that splits on all non-alphanumerical characters. The score column shows the heuristic value used for this proposition type when ranking candidate propositions for a folder.

<table>
<thead>
<tr>
<th>Proposition Type</th>
<th>Generator</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>From(e)</td>
<td>Full email addresses in sender field.</td>
<td>5</td>
</tr>
<tr>
<td>InFrom(e)</td>
<td>Tokens in sender.</td>
<td>4</td>
</tr>
<tr>
<td>To(e)</td>
<td>Full addresses in receiver, cc and bcc fields.</td>
<td>5</td>
</tr>
<tr>
<td>InTo(e)</td>
<td>Tokens in receiver, cc and bcc fields.</td>
<td>4</td>
</tr>
<tr>
<td>InFromOrTo(e)</td>
<td>Tokens in sender, receiver, cc and bcc fields.</td>
<td>3</td>
</tr>
<tr>
<td>SubjectContains(e)</td>
<td>Tokens in subject field and name of the folder.</td>
<td>5</td>
</tr>
<tr>
<td>BodyContains(e)</td>
<td>Tokens in email body and name of the folder.</td>
<td>1</td>
</tr>
<tr>
<td>InSubjectOrBody(e)</td>
<td>Tokens in subject and email body, and name of the folder.</td>
<td>2</td>
</tr>
</tbody>
</table>

Note that this means \(u(c_i) \setminus \bigcup_{j \neq i} u(c_j)\) is not empty, otherwise we could remove \(c_i\) and obtain a shorter rule for the folder.
Both steps follow the same pattern of first trying to replace existing propositions (lines 15–23 and 36–43) and only adding disjuncts (generalize) or conjuncts (specialize) if replacement fails (lines 24–28 and 45–51). Candidates for replacement or addition are greedily generated from the power sets \( \mathcal{P} \) and \( \mathcal{S} \) (version 3.8.7) on an Intel Core i7 processor (base at 1.8 GHz) and a K80 GPU, a 64-bit operating system, and 32 GB RAM.

### 4.2 Datasets

The experiments are carried out on Python (version 3.8.7) on an Intel Core i7 processor (base at 1.8 GHz) and a K80 GPU, a 64-bit operating system, and 32 GB RAM.

### 4.3 Suppression Rules

To mitigate low-confidence predictions, for each incoming email, we explicitly predict whether a folder assignment should be suppressed or not. Our suppression model uses a linear combination of five features with sigmoid activation to make predictions. The model features are rule length, number of consecutive correct predictions by the rule, running accuracy for the folder, average running accuracy of specific disjuncts that the mail satisfied, and folder size. Weights are optimized using gradient descent. Suppression is related to learning with rejection [3], but in our setting we employ the model described to suppress predictions made by rules. Algorithm 3 describes how EmFore predicts folder assignment for a new mail.

### 4.4 Evaluation Setup

In this section, we describe the datasets and setup used to evaluate EmFore, and the various existing and adapted baseline systems.

#### 4.4.1 System Specifications

The experiments and studies have been carried out using Python (version 3.8.7) on an Intel Core i7 processor (base at 1.8 GHz) and a K80 GPU, a 64-bit operating system, and 32 GB RAM.

#### 4.4.2 Datasets

We use two datasets in our evaluation. The first (and most popular) is the Enron dataset [25]. After removing duplicates and outgoing folders, we are left with 46,096 emails in 2,612 folders across 150 users. The second is the Avocado dataset [35] with 88,172 emails in 3,423 folders across 277 users after similar processing.
Previous work often evaluates on the Enron–Bekkerman (EB) subset, which consists of seven users in the Enron corpus with a high number of mails. We report one result on this subset for completeness, but our approach is less reliant on having a large number of examples and does not discriminate on number of mails.

4.3 Setup
We evaluate EmFore in both the offline setup used in previous work [10, 17] and an online setup. For both cases, all mails are ordered chronologically for each user. In the offline setup, $k$% of mails are used for training and either all remaining mails or the next 10% of mails is used for testing. In the former case, $k$ is set to different values (50, 60 and 90) and the results are averaged to get a single score. The latter setup was introduced with ALECSA [17] to take concept drift into account during evaluation.

Our online setup is aimed at more accurately reflecting how a user would experience the system. For every mail, the system either correctly places it in a folder, incorrectly places it in a folder or neutrally leaves it in the inbox whereas it should have been in a folder. The proportion of correct decisions (+) and incorrect decisions (−) are both recorded—everything else is a neutral decision. Whenever an incorrect or neutral decision is made, the system is allowed to retrain with the groundtruth label. To avoid large folder bias, we average results first by folder and then across folders.

4.4 Baselines
We compare EmFore against a mixture of published systems that perform folder classification and custom baselines that use popular NLP and classification approaches that we adapted for this task.

- Decision trees [18], Support Vector Machines [7], Naive Bayes [31] and Winnow [4] were among early methods applied to the problem of email folder classification.
- ALECSA [17] and ABC-DynF [10] are recent systems specifically designed for email folder classification. Unlike EmFore, these systems do not generate rules for folders. We implement these systems as described by the authors in [17] and [10] respectively. For ALECSA, since the consultation cost, reward and punishment hyper-parameters used by the authors are not available we perform a grid search and report the best performance across all parameter values tested. The systems are described in more detail in Section 7.
- POPPER is a state-of-the-art inductive logic programming system [16]. We use Popper to learn a rule over our predicates for each folder. Popper shares the ideas of generalization and specialization that we extend in EmFore for online learning.
- Incremental Decision trees [42] are decision trees that are incrementally learned over sequential batches of data. Similar to EmFore it is also an online learning system.
- Constrained clustering is a semi-supervised clustering technique which uses labelled data to generate linkage constraints that later guide the clustering of unlabelled samples. We use COP-KMeans [44] which is a popular constrained clustering technique based on K-Means [29].
- SentenceBERT is a popular sentence embedding model trained for multiple downstream language tasks [37]. We add a classification layer on top and fine-tune it end-to-end.
- KNN-BERT is a recent method that uses a KNN classifier to optimize BERT embeddings for text classification using an end-to-end model [27]. We fine-tune the KNN-BERT model for our classification task. We set $\phi = 0.5$ balancing the linear and KNN component in the final prediction. We test with 3, 5 and 10 neighbours and report the best performance.
- Contrastive learning optimizes the separation between examples of different classes. We implement the contrastive loss as described in [41] and train BERT embeddings followed by classification. Positive samples are emails from a folder and negative samples are taken from other folders.
- T5 [36] is an encoder-decoder transformer pre-trained on language. We fine-tune T5 to generate the target folder name given the email header, body and available folder names.
- GPT-3.5 [8] is state-of-the-art language model. Like T5, we prompt the model using the header, body and available folder names and generate the target folder name.

4.5 Model Training
Optimizing weights for our method (ranking and suppression) and pre-training of supervised baselines needs data. Training on other users from the same dataset can bias results, as emails in the two corpora are from within single companies. We therefore train on one dataset and evaluate on the other.

To generate data for optimizing suppression weights, we run the online learning setup without suppression to obtain (rule, correctness) pairs. Since EmFore still performs well without suppression, this process is repeated multiple times for shuffled folders and a balanced set of correct and incorrect decisions is sampled. Neutral decisions are counted as incorrect.

In the offline setting, all baselines are trained on the seen emails for each user and evaluated on the rest of the emails. In the online setting, for the neural baselines, we continue training the models with new data. ALECSA and ABCDynF already define an update method over batches. For all other baselines, in the online setup, we retrain from scratch after each incorrect email classification.

5 RESULTS
We perform an extensive evaluation to answer the following research questions.

Q1. Can EmFore quickly and accurately learn folder rules?
Q2. Can EmFore learn rules for folders with diverse emails?
Q3. Can EmFore suppression reduce false positives?
Q4. How does expressivity of rules affect performance?
Q5. How many emails need to be stored per folder to update rules without sacrificing performance?

5.1 Performance (Q1)
Our symbolic learner makes more correct decisions (+) and fewer incorrect decisions (−) than all baselines in the online evaluation, as shown in Table 2. EmFore obtains a higher correct decision rate than baselines in the offline evaluation used in previous work [10, 17]. We highlight how EmFore’s design enables this performance.

EmFore ensures that rules remain consistent on all historical emails. This consistency, in combination with suppression, keeps the incorrect decision rate low, even when learning rules for noisy
Table 2: System comparison. Rules denotes if a system can yield symbolic folder rules. In the online setup, EmFore makes more correct decisions (+) and fewer incorrect decisions (−) than baselines. For completeness, we also show correct decision rate for five offline setups aggregated over Enron and Avocado datasets, as done in prior work [10, 17]. EmFore outperforms all baselines in the offline setup as well. Asterisks (*) denote per-user timeout after five minutes.

<table>
<thead>
<tr>
<th>System Description</th>
<th>Online (+ and −)</th>
<th>Offline (+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Prior Work</td>
<td>Rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision Tree</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Incremental Decision Tree</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Moving Winnow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Popper</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Constrained Clustering</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sentence BERT + Classification</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>T5</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BERT K-Nearest Neighbours</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Contrastive Learning</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ABC-DynF</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Alecsa</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EmFore</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

folders. By first performing a replacement step to bias towards short rules and ranking candidate propositions, EmFore prevents heavy overfitting on these historical assignments. Approaches with strong generalization (like neural networks) or that are too greedy (like decision trees) fail to keep the incorrect decision rate low.

Because EmFore is designed to favour precision over recall, the ability to learn from each mistake causes the correct decision rate to be higher than baselines. Focusing on each mistake also helps to tackle concept drift, an example of which is shown in Figure 4. When the user decides to expand the scope of a folder from FedEx to delivery in general, a single email is sufficient for EmFore to update its rule. Figure 5 shows the cumulative correct decision rate (+) when updating after 1, 2, 5 or 10 emails. Especially when the number of mails is small and the user is deciding the scope of a folder, updating the rule more frequently significantly impacts performance of future classifications.

EmFore is fast enough to carry out such rule updates after every iteration. Figure 6 shows the learning time as a function of the number of mails for EmFore, the best neural baseline and the best existing method. Since EmFore is an online system, we show both the cumulative time and the time taken at each iteration. Updating the rule only takes a fraction of a second—four orders of magnitude faster than Alecsa. Even cumulatively, EmFore is an order of magnitude faster than Alecsa.

Table 3 shows some examples of the simple and interpretable rules learned by EmFore for the user arnold-j from the Enron corpus.

5.2 Variety in Emails (Q2)

One advantage of neural methods is the ability to make semantic classifications, where the user has a clear intent but there is no rule that captures it. As an example, KNN-BERT achieves 68% folder accuracy on folders named “personal” compared to 61% (EmFore) and 54% (Alecsa). The average number of propositions for these
Figure 5: Cumulative CDR of EmFORE when updating after 1, 2, 5 or 10 mails on the Enron dataset. Updating at every iteration allows EmFORE to resolve concept drift when detected.

Figure 6: Learning time for increasing number of mails. For EmFORE we show both per-update and cumulative rule learning time. We plot the first 500 emails for better visualization, however the same trend persists across additional emails.

Figure 7: Correct decision rate and Silhouette score by length of EmFORE rule. Folders with lower performance and Silhouette score have longer rules.

We estimate the variety of emails in folders by treating folders as clusters within a user and computing their Silhouette score [39]. We define similarity between two emails as the Hamming distance between all propositions generated for them. Figure 7b shows how the Silhouette score decreases as learned rules get longer. However, Figure 8 shows that EmFORE is more robust on folders of different quality compared to the next-best baseline for each folder —scoring between 10 and 15 absolute percentage points higher within the Silhouette score range that contains most of the folders.

Figure 8: Comparing EmFORE and the best baseline for each folder as a function of Silhouette score. Performance is comparable at the edges, but EmFORE performs better overall.

Figure 9 shows the correct decision rate for a clean (williams-w3) and noisy (beck-s) user based on the Silhouette score. Each red line represents a new folder being created. For the clean user, EmFORE quickly learns a good representation. For the noisy user, EmFORE quickly learns reasonable rules for some folders, but as the user adds folders without consistent topics, performance decreases.

5.3 Suppression (Q3)

To reduce false positives it is important that EmFORE can suppress unreliable classifications. In addition to our trained suppression model, we also evaluate the following suppression strategies.

- No suppression.
- Only predict with rules shorter than a specified length.
- Only predict with rules for which the previous $k$ predictions were correct.
- Neural suppression, which encodes the incoming email and 10 last emails with T5, combines them with cross-attention, concatenates the manual features and passes the result through a linear layer with sigmoid activation.
Table 4 shows that training a suppression model based on features of the rules clearly minimizes the number of incorrect moves (−) without significantly sacrificing correct ones (+). Allowing only one proposition per rule reduces the proportion of incorrect moves (−4.5%) but affects the correct moves even more (−18%). Waiting for correct classifications before moving a mail shows even more drastic reduction in incorrect moves but with a significant drop in correct decision rate. Neural suppression model reduces the proportion of incorrect moves (−0.3%), but it relies on a huge model (60M parameters) which makes inference 20 times slower than EmFORE.

Table 4: Average suppression time, correct (+) and incorrect (−) decision rates for different rule suppression strategies. EmFORE’s suppression model barely impacts correct decisions and is comparable to neural while significantly faster.

<table>
<thead>
<tr>
<th>Suppression Strategy</th>
<th>Time</th>
<th>Enron</th>
<th>Avocado</th>
</tr>
</thead>
<tbody>
<tr>
<td>No suppression</td>
<td>0ms</td>
<td>83.7</td>
<td>7.2</td>
</tr>
<tr>
<td>Rule length = 1</td>
<td>4ms</td>
<td>64.8</td>
<td>2.7</td>
</tr>
<tr>
<td>Rule length ≤ 3</td>
<td>4ms</td>
<td>76.1</td>
<td>61.1</td>
</tr>
<tr>
<td>Last email correct</td>
<td>12ms</td>
<td>70.5</td>
<td>4.3</td>
</tr>
<tr>
<td>Last two emails correct</td>
<td>12ms</td>
<td>66.5</td>
<td>0.8</td>
</tr>
<tr>
<td>Neural suppression</td>
<td>576ms</td>
<td>83.1</td>
<td>3.9</td>
</tr>
<tr>
<td>EmFORE</td>
<td>25ms</td>
<td>83.4</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Figure 10 shows how the correct, neutral and incorrect decision rates evolve as more mails are seen for different suppression methods. We see that EmFORE predicts neutral (No Move) for relatively fewer emails (7.3%) and thus does not sacrifice coverage to reduce false positives (Wrong Move) unlike other variations. Requiring the last move to be correct and only allowing rules with three or fewer propositions both keep the incorrect decision rate low, but also sacrifices a lot of correct decisions. Our suppression model decreases the incorrect decision rate, without reducing correct decisions or blowing up the inference time. Suppression based on last moves highlights that EmFORE makes effective use of each mistake, as the correct decision rate without suppression is much higher (+13%).

5.4 Expressivity (Q4)

different email clients support different rules, all of which can be translated into DNF over a predicate space. Table 5 shows the correct (+) and incorrect (−) decision rates of EmFORE with restricted grammars and examples of clients that support these rule sets.

Negatives are rarely required and not allowing them does not substantially impact performance (−1.2%). Not using conjunctions causes worse propositions to be used as disjuncts (−3.5%). When not allowing disjunctions, EmFORE becomes worse at coping with folders with a wider scope (−7%). In practice, all clients support disjunctions by creating multiple rules for each folder.

5.5 Information Retention (Q5)

We investigate the effect of storing a subset of the folder’s mail. In practice, email clients do not store all mails locally because of space constraints, and only a subset is available locally at any instant. For EmFORE to be deployed in clients, it needs to maintain performance without access to the entire history while updating a rule.

For this experiment, we compare EmFORE against the best baseline (ALECSA) and restrict both systems to only have access to the latest k emails. Figure 11 shows how the correct and neutral decision rates evolve as a function of the number of retained emails. We find that EmFORE consistently outperforms ALECSA by 10 absolute percentage points in correct decision rate. Additionally, EmFORE sees diminishing returns from storing more emails faster (20 versus 40). These results show that EmFORE can effectively update rules without accessing the entire mail history.

6 DISCUSSION

Unfortunately, much of the research on email classification has been carried out on private industrial datasets [6]. Our experiments are carried out on two public datasets: Enron and Avocado, which to the best of our knowledge remain the only public email corpora actively used in research. As a result, performance on corpora that have substantially different characteristics may be different.

As common in programming-by-example (PBE), EmFORE assumes the user provides accurate examples from which to learn. Prior work on neural methods for PBE have explored learning from noisy examples [19]. More recently, weighted finite-tree automata have been applied to (symbolically) learn programs from noisy examples [24]. Extending these ideas and evaluating them in the context of email classification rule learning remains future work.

The approach underlying EmFORE may be applicable to other domains. Specifically, domains that require (1) simple rules learned in an online fashion and (2) rules can be generated based on simple syntactic predicates over meta-data or content. Exploring such domains (e.g. document/folder classification) remains future work.

7 RELATED WORK

RIPPER [13], which is based on a greedy keyword search, was the first text classifier evaluated on email folder classification. Later, many popular classification methods were applied to this domain: Naive Bayes, support vector machines and Winnow based techniques [5]; neural networks [12]; random forests and ensembles [26]. More recent work includes ALECSA [17] and ABC-DynF [10]. ALECSA uses an attention control mechanism to determine which structural properties of emails should be used to assign emails to folders. ABC-DynF uses Iterative Bayes [21] to update the weights over a dynamic feature space when receiving batches of emails.
Past work in Inductive Logic Programming (ILP) have explored the ideas of generalization and specialization used by EmFore to represent hypotheses and data using first-order logic [33]. [15] uses n-grams to learn first-order rules for classification. [14] extends this idea by making the rules more readable. Incremental learning has been studied in ILP as Theory Revision [33], which requires a full history pass and become progressively more expensive. Like ILP systems, EmFore uses symbolic predicates over emails that are interpretable and can be inspected by the user. EmFore shares motivation and ideas from these systems but there are two key differences. First, EmFore is an online learning system that uses a novel state abstraction for efficient updates to the ruleset after each new email. Second, EmFore uses predicate ranking and suppression that allow it to update efficiently in an online setting.

Related tasks like category prediction, spam detection and priority modeling have used graph neural networks [11], deep belief networks [40] and word embeddings [2] trained on corpora of labelled mails for classifying new incoming emails. Another related area of work is to group emails regarding their topic of discussion into email conversation threads [12]. These systems classify mails into predefined categories that are common for all users and known ahead of time. Unlike these problems, EmFore handles dynamic categories unique to each user that the user can modify over time.

EmFore: Online Learning of Email Folder Classification Rules CIKM ’23, October 21–25, 2023, Birmingham, United Kingdom

8 CONCLUSION

We introduce EmFore, an online system for learning email folder classification rules by observation. Unlike prior machine learning approaches that treat this as a pure classification task, EmFore generates symbolic rules that are supported by modern email clients. Unlike prior inductive logic programming (ILP) approaches, EmFore learns rules in an online fashion by using an abstraction of folder states to efficiently update rules. To mitigate low confidence predictions, EmFore uses a suppression model. We carry out extensive experiments on two datasets and show that EmFore outperforms 14 baselines that represent state-of-the-art email classification systems, machine learning approaches, incremental learning approaches, ILP approaches, and neural models. Our results show EmFore learns orders of magnitude faster than competitive baselines, while producing rules that more accurately classify emails.

9 ACKNOWLEDGEMENTS

We thank Siba Panigrahi for initial investigations on these datasets.