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 folder assignment systems, incremental machine learning, 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

ACM Reference Format:
Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, and Gust Verbruggen. 2023. EmFore: Online Learning of Email Folder Classification Rules. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3583780.3614863

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$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 InFrom("straw") ∨ InTo("straw") 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 $\mathcal{R}(m_i) = f_i$. 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 $\mathcal{R}(m_\star) = f$ for a new email $m_\star$ is wrong, as indicated by the user moving the email to folder $f^\star$, we update the rule to be consistent with all previous emails and the new email. We introduce three components for doing so: a state $\mathcal{S}$ 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 supported by different popular email clients.

Example 2. *Some propositions for an email with receiver line "strawbale@crest.org, absteen@dakotacom.net" are*

To("strawbale@crest.org")	*To("absteen@dakotacom.net")*
InTo("strawbale")	*InTo("crest")*
...	
InFromOrTo("strawbale")	*InFromOrTo("dakotacom")*

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

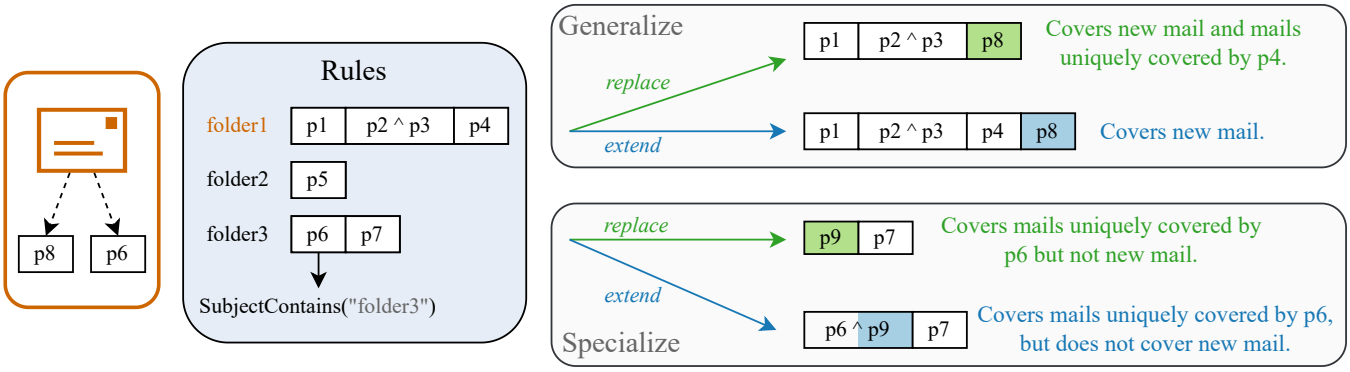


Figure 1: Generalization and specialization steps for a mail that should satisfy folder1 but instead satisfies folder3. First, we try to replace propositions with candidates from the state such that folder1 satisfies the mail (generalize) and folder3 does not (specialize). If no replacements are found,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

proposition. 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).*

InTo("strawbale") > InFromOrTo("strawbale") > To(" strawbale@crest.org") > InTo("crest") > To(" absteen@dakotacom.net") > InFromOrTo("dakotacom")

Whenever an email (m_i, f_i) comes in, propositions P_i 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 $c_1 \vee c_2$ be a rule with $c_1 = p_1 \wedge p_2$ 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 $u(c_i)$.

Table 1: Proposition templates supported by EmFore, inspired by six popular email clients. A generator substitutes e 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.

Proposition	Generator	Score
From(e)	Full email addresses in *sender* field.	5
InFrom(e)	Tokens in *sender*.	4
To(e)	Full addresses in *receiver*, *cc* and *bcc* fields.	5
InTo(e)	Tokens in *receiver*, *cc* and *bcc* fields.	4
InFromOrTo(e)	Tokens in *sender*, *receiver*, *cc* and *bcc* fields.	3
SubjectContains(e)	Tokens in *subject* field and name of the folder.	5
BodyContains(e)	Tokens in email body and name of the folder.	1
InSubjectOrBody(e)	Tokens in *subject* and email body, and name of the folder.	2

Note that this means $u(c_i) \setminus \cup_{j \neq i} u(c_j)$ is not empty, otherwise we could remove c_i and obtain a shorter rule for the folder.

3.3 Updating Rules

When a new email $(m_\star, f^\star)$ comes in, each of the rules $R_f \in \mathcal{R}$ may be updated. The full algorithm is shown in Figure 2. If $m_\star \not\models R_{f^\star}$, meaning the email is not covered by the current rule for folder $f^\star$, then it requires generalization (the `cover` function, lines 12–28). Any rule R_f with $f \neq f^\star$ and $m_\star \models R_f$, 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.

```
1  function update(Map[Folder → Set[Predicate]] S,
2                  Map[Folder → Set[Set[Predicate]]] R,
3                  Set[Predicate] P_all,
4                  Mail m,
5                  Folder f):
6     if m ⊭ R_f:
7        R_f ← cover(S_f, R_f, m, f)
8     foreach (f' → P) in S:
9        if m ⊨ R_f':
10          R_f' ← uncover(S_f', R_f', P_all, m, f')
11
12 function cover(Set[Predicate] P,
13                Set[Set[Predicate]] R,
14                Mail m, Folder f):
15    // candidate replacements that cover m
16    cands ← {(p, c) | p ∈ 2^P, c ∈ R, m ⊨ p,
17                      ∀ m' ∈ u(c): m' ⊨ p}
18    // if any are found, apply shortest one
19    // that is ranked highest
20    if cands is not empty:
21       short ← {(p, c) | |p| = min_(p, c) ∈ cands |p|}
22       p, c  ← argmax_(p, c) ∈ short rank(p, P \ c, R, f)
23       return (R \ Set([c])) ∪ Set([p])
24    // add the best shortest new disjunct
25    preds ← {p | p ∈ 2^P, m ⊨ p }
26    short ← {p | |p| = min_p∈preds |p|}
27    p ← argmax_p∈ short rank(p, P, R, f)
28    return R ∪ Set([p])
29
30 function uncover(Set[Predicate] P,
31                  Set[Set[Predicate]] R,
32                  Set[Predicate] P_all,
33                  Mail m, Folder f):
34    foreach (c ← { c | c ∈R, m ⊨ c})
35        // candidate replacements that do not cover m
36        cands ← {p | p ∈ 2^P, m ⊭ p,
37                           ∀ m' ∈ u(c): m' ⊨ p}
38        // if any are found, apply shortest one
39        // that is ranked highest
40        if cands is not empty:
41            short ← {p | |p| = min_p∈ cands |p|}
42            p ← argmax_p ∈ short rank(p, P \ c, R, f)
43            R ← (R \ Set([c])) ∪ Set([p])
44        else
45            // add from rejected predicates
46            cands ← {p | p ∈ 2^P_all,
47                                m ⊭ p,
48                                ∀ m' ∈ u(c): m' ⊨ p}
49            short ← {p | |p| = min_p∈ cands |p|}
50            p ← argmax_p∈ short rank(p, P \ c, R, f)
51            R ← (R \ Set([c])) ∪ Set([c ∪ p])
```

Figure 2: Rule update. In practice, the shortest elements are lazily generated from the power sets 2^P and $2^{P_{\text{all}}}$. For brevity we use a set of predicates to denote a conjunction and a set of set of predicates to denote a disjunction over conjunctions.

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 selected from the ranking stored in the state. Since replacement guarantees a rule with minimal disjuncts, for generalization of a disjunct, we only need to consider candidates p for disjunct c if $m \models p$ for all $m \in u(c)$ and $m_\star \models p$ (line 16) or just $m_\star \models p$ (line 25). During specialization of a conjunct c, we only need to consider p such that $m \models p$ for all $m \in u(c)$ and $m_\star \not\models p$ (lines 36 and 46).

Example 4. *An overview of generalization and specialization is shown in Figure 1. The new mail is assigned to folder3, but it should have been in folder1, as detected by the user correcting this mistake. Both replace and extend steps are shown, but we only extend if no replacement is found.*

```
1  function predict(Map[Folder → Set[Predicate]] S,
2                   Map[Folder → Set[Set[Predicate]]] R,
3                   Mail m):
4     cov ← []
5     foreach (f → P) in S:
6        if m ⊨ R_f:
7           cov ← cov ∪ m
8     if len(cov) = 1 and not suppress(S, cov[0], m)
9         return f
10    return null
```

Figure 3: EmFore prediction. If `predict` returns `null`, the email remains in the default (inbox) folder.

While updating rules, when possible, EmFore ensures that they remain consistent with all past predictions. If consistency cannot be guaranteed, for example, because of concept drift or the user making a mistake, the rule is not updated. Any risk of *overfitting* on past predictions is by design—this reduces the number of false positives, and the rule can be generalized on the next email that is moved into the folder. The update algorithm is linear in the number of folders (f) and rule length (R) and exponential in the number of predicates (P). Theoretically, this can exhibit exponential complexity but since the power set construction is done lazily the observed complexity is much lower as indicated in our latency experiments (Section-5.1).

3.4 Suppressing 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 EVALUATION SETUP

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

4.1 System Specifications

All 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.

4.2 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 (30, 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 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.

System Description			Online (+ and −)				Offline (+)				
Name	Prior Work	Rules	Enron		Avocado		Train/test			30/60/90 + 10	
			+	−	+	−	10/90	50/50	90/10	All	EB
Decision Tree	Yes	Yes	63.7	13.6	67.2	12.6	49.9	60.6	73.9	79.6	70.0
Incremental Decision Tree	Yes	Yes	64.3	10.2	68.8	11.3	49.6	60.4	73.7	79.3	69.8
Support Vector Machine	Yes	No	64.6	13.4	67.5	12.5	50.2	61.5	74.0	80.3	72.5
Naive Bayes	Yes	Yes	58.6	14.6	61.8	13.6	46.3	55.4	68.7	76.8	68.3
Moving Winnow	Yes	Yes	61.8	14.0	64.9	13.0	48.6	58.5	73.8	79.3	69.5
Popper	No	Yes	*	*	*	*	52.8	62.8	76.2	81.5	70.3
Constrained Clustering	No	No	60.1	14.3	64.3	13.1	47.2	56.3	72.5	79.5	69.7
Sentence BERT + Classification	No	No	60.4	14.3	62.7	13.4	46.9	57.8	70.7	77.6	68.8
T5	No	No	61.9	14.0	65.3	12.9	47.1	59.6	75.2	78.3	69.7
GPT-3.5	No	No	*	*	*	*	61.7	64.0	65.3	75.6	68.1
BERT K-Nearest Neighbours	No	No	67.4	12.8	71.7	11.7	53.4	64.8	77.1	80.6	72.3
Contrastive Learning	No	No	65.1	13.3	69.8	12.1	50.6	61.9	76.1	80.4	71.6
ABC-DynF	Yes	No	74.2	11.6	78.1	10.7	55.3	67.7	76.7	82.3	70.6
Alecsa	Yes	No	74.8	11.4	78.4	10.4	55.1	68.2	77.3	83.4	74.1
EmFore		**Yes**	**83.4**	**4.2**	**87.1**	**3.7**	**66.7**	**78.3**	**84.1**	**87.0**	**78.7**

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 relearning in 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. Rule interpretability is crucial for deployment because users should be able to easily verify and edit folder rules generated by EmFore.

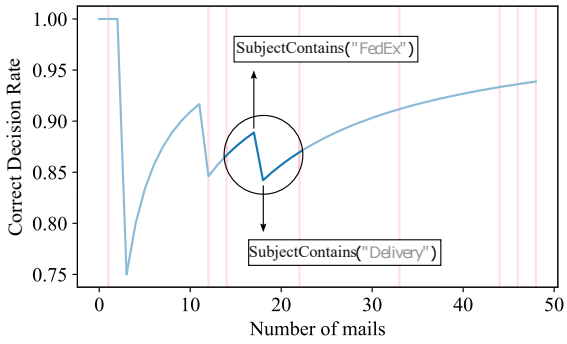


Figure 4: Concept drift for Enron user arnold-j, who initially only added FedEx mails to the *fedex* folder but later added everything related to delivery. EmFore adapts to this change after a single email.

Table 3: Examples of simple, inspectable, and understandable EmFore rules for Enron user *"arnold-j"*'s first 5 folders.

Folder	EmFore Rule
Avaya	SubjectContains("Avaya")
Compaq	InFrom("Compaq") ∨ SubjectContains("Compaq")
EBS	InFromOrTo("EBS")
Airline	InFrom("Continental") ∧ SubjectContains("Flight")
Colleen	InFromOrTo("colleen")

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

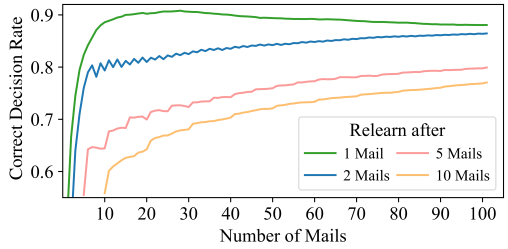


Figure 5: Cumulative CDR of EmFore when updating after 1, 2, 5 or 10 mails on the Enron dataset. Updating at every iteration allows EmFore to resolve concept drift when detected.

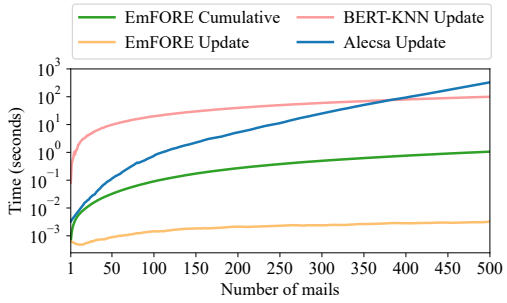
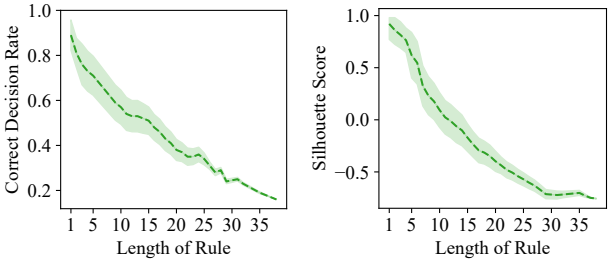


Figure 6: Learning time for increasing number of mails. For EmFore we show both per-update and cumulative rule learning time. We plot the first 500 emails for better visualization, however the same trend persists across additional emails.

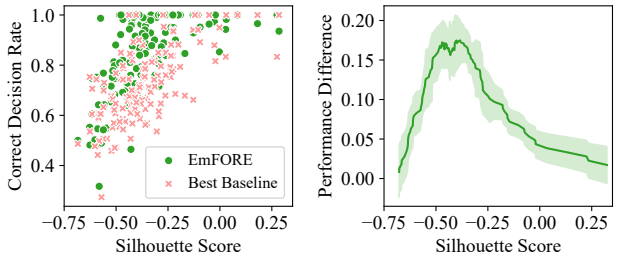
folders is 13, which provides some indication that EmFore is not capturing the correct intent. Figure 7a shows that correct decision rate deteriorates with the number of propositions in the rule.



(a) Correct decision rate **(b) Silhouette score**

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.



(a) Correct decision rate (CDR). **(b) Difference in CDR.**

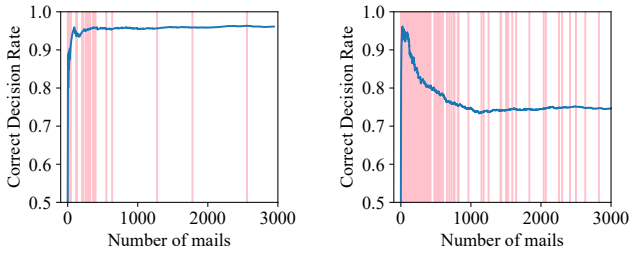
Figure 8: Comparing EmFore and the best baseline for each folder as a function of Silhouette score. Performance is comparable at the edges, but EmFore performs better overall.

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.



(a) Clean user (williams-w3) **(b) Noisy user (beck-s)**

Figure 9: Cumulative correct decision rate of EmFore for a noisy and clean user (based on Silhouette score) from the Enron-Bekkerman dataset. There is a clear performance gap between clean and noisy users.

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.

Suppression Strategy	Time	Enron		Avocado	
		+	−	+	−
No suppression	0ms	83.7	7.2	87.5	6.6
Rule length = 1	4ms	64.8	2.7	68.5	1.8
Rule length ≤ 3	4ms	76.9	3.6	81.1	3.0
Last email correct	12ms	70.5	4.3	74.1	3.9
Last two emails correct	12ms	66.5	0.8	70.3	0.6
Neural suppression	576ms	83.1	3.9	86.8	3.5
EmFore	25ms	83.4	4.2	87.1	3.7

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.

Table 5: Correct (+) and incorrect (-) decision rates for EmFore with different rule grammars along with popular mail clients that use these grammars. Full grammar performs the best.

Grammar	Client	Enron		Avocado	
		+	−	+	−
Full Grammar	Outlook	83.4	4.2	87.1	3.7
No Negatives	Thunderbird	82.1	8.0	86.0	5.6
No Conjunction	Gmail	79.7	9.4	83.7	7.0
No Disjunction		76.3	11.5	79.8	9.4

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

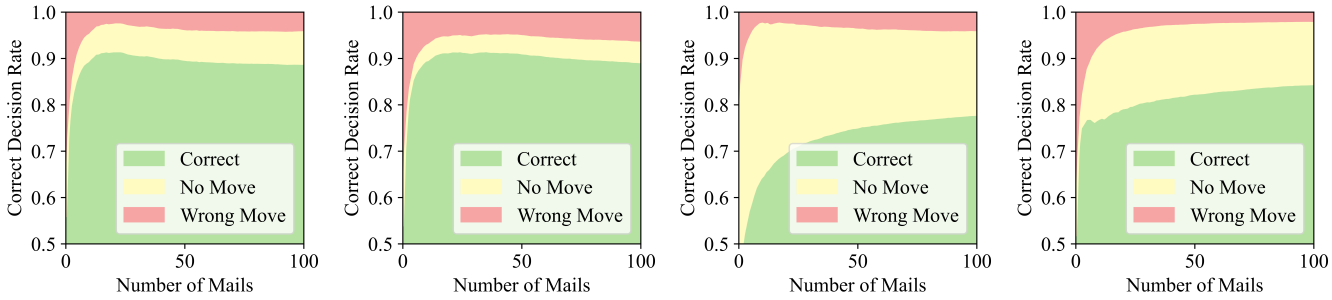
Unfortunately, much of the research on email classification has been carried out on private industrial datasets [6]. Our experiments are carried out on two public datasets: Enron and Avocado, which to the best of our knowledge remain the only public email corpora actively used in research. As a result,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

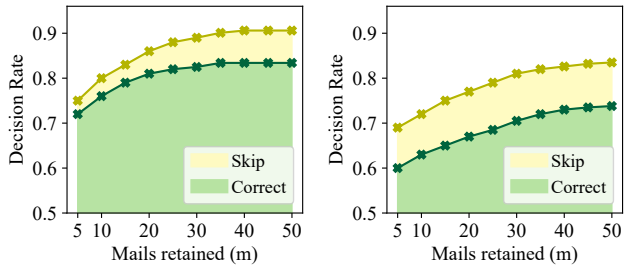
7 RELATED WORK

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: Naïve 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.



(a) EmFore suppression (b) No Suppression (c) Last move correct (d) Rule length 3

Figure 10: Evolution of correct, neutral and incorrect decision rates for different suppression methods. EmFore suppression keeps the number of correct decisions high and reduces the number false positives. Requiring a correct classification (c) and limiting the rule length (d) achieves even fewer false positives, but they achieve significantly fewer correct predictions.



(a) EmFore (b) Alecsa

Figure 11: Correct (+) decision rate keeping m most recent mails. EmFore maintains performance retaining only 20.

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.

Grbovic et al. [22] explore the possibility of automatically classifying emails into a limited set of *latent* folders, rather than user-defined folders. Di Castro et al. [20] learn to predict the next action (read, delete, reply, etc) for an email upon receipt. EmFore is explicitly designed to learn user-defined folder rules, which in turn may be complementary as features for next-action prediction.

Surveys in email folder classification [34, 45] have pointed out challenges such as false positives, concept drift, the need for an evaluation that more closely mirrors user experience, lack of neural baselines, and onerous classification time. We designed EmFore and its evaluation to address these concerns. Bendersky et al. [6] surveyed recent work in search and discovery in personal email; EmFore's PBE-approach complements the work detailed there.

Transformer-based models [43] have led to state-of-the-art performance on a broad set of natural language processing (NLP) tasks. We adapt T5, KNN-BERT [27] and contrastive learning [41] on top of BERT for the task of email folder classification and use these as baselines to evaluate EmFore.

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.

REFERENCES

[1] Tarfah Alrashed, Chia-Jung Lee, Peter Bailey, Christopher Lin, Milad Shokouhi, and Susan Dumais. 2019. Evaluating user actions as a proxy for email significance. In *The World Wide Web Conference*. Association for Computing Machinery, New York, NY, USA, 26–36.

[2] Eman M. Bahgat, Sherine Rady, Walaa Gad, and Ibrahim F. Moawad. 2018. Efficient email classification approach based on semantic methods. *Ain Shams Engineering Journal* 9, 4 (2018), 3259–3269. https://doi.org/10.1016/j.asej.2018.06.001

[3] Peter L Bartlett and Marten H Wegkamp. 2008. Classification with a Reject Option using a Hinge Loss. *Journal of Machine Learning Research* 9, 8 (2008), 1823–1840.

[4] R. Bekkerman, A. McCallum, and G. Huang. 2004. Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora. CIIR Technical Report, IR-418, University of Massachusetts, Amherst, USA.

[5] Ron Bekkerman, Andrew McCallum, and Gary B. Huang. 2005. *Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora*. Technical Report. University of Massachusetts Amherst.

[6] Michael Bendersky, Xuanhui Wang, Marc Najork, and Donald Metzler. 2022. Search and discovery in personal email collections. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, New York, NY, USA, 1617–1619.

[7] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*. Association for Computing Machinery, New York, NY, USA, 144–152.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., USA, 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[9] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by time or by relevance? Revisiting email search. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 283–292.

[10] José M Carmona-Cejudo, Gladys Castillo, Manuel Baena-García, and Rafael Morales-Bueno. 2013. A comparative study on feature selection and adaptive strategies for email foldering using the ABC-DynF framework. *Knowledge-Based Systems* 46 (2013), 81–94.

[11] Sharma Chakravarthy, Aravind Venkatachalam, and Aditya Telang. 2010. A Graph-Based Approach for Multi-folder Email Classification. In *2010 IEEE International Conference on Data Mining*. IEEE Computer Society, USA, 78–87. https://doi.org/10.1109/ICDM.2010.55

[12] James Clark, Irena Koprinska, and Josiah Poon. 2003. A neural network based approach to automated e-mail classification. In *Proceedings IEEE/WIC international conference on web intelligence (WI 2003)*. IEEE, IEEE Computer Society, USA, 702–705.

[13] William W Cohen et al. 1996. Learning rules that classify e-mail. In *AAAI spring symposium on machine learning in information access*, Vol. 18. Stanford, CA, AAAI, USA, 25.

[14] Elisabeth Crawford, Judy Kay, and Eric McCreath. 2002. Automatic Induction of Rules for e-mail Classification. *Elsevier* 1, 1 (04 2002).

[15] Elisabeth Crawford, Judy Kay, and Eric McCreath. 2002. IEMS - The Intelligent Email Sorter. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML '02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 83–90.

[16] Andrew Cropper and Rolf Morel. 2021. Learning programs by learning from failures. *Machine Learning* 110, 4 (2021), 801–856.

[17] Mostafa Dehghani, Azadeh Shakery, and Maryam S Mirian. 2016. Alecsa: attentive learning for email categorization using structural aspects. *Knowledge-Based Systems* 98 (2016), 44–54.

[18] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE transactions on pattern analysis and machine intelligence* 19, 4 (1997), 380–393.

[19] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*. PMLR, Association for Computing Machinery, New York, NY, USA, 990–998.

[20] Dotan Di Castro, Zohar Karnin, Liane Lewin-Eytan, and Yoelle Maarek. 2016. You've got mail, and here is what you could do with it! analyzing and predicting actions on email messages. In *The 9th ACM conference on web search and data mining*. Association for Computing Machinery, New York, USA, 307–316.

[21] Joao Gama. 2003. Iterative bayes. *Theoretical Computer Science* 292, 2 (2003), 417–430.

[22] Mihajlo Grbovic, Guy Halawi, Zohar Karnin, and Yoelle Maarek. 2014. How many folders do you really need? classifying email into a handful of categories. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 869–878.

[23] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[24] Shivam Handa and Martin C Rinard. 2020. Inductive program synthesis over noisy data. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 87–98.

[25] Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*. Springer, Springer, Berlin, Heidelberg, 217–226.

[26] Irena Koprinska, Josiah Poon, James Clark, and Jason Chan. 2007. Learning to classify e-mail. *Information Sciences* 177, 10 (2007), 2167–2187.

[27] Linyang Li, Demin Song, Ruotian Ma, Xipeng Qiu, and Xuanjing Huang. 2021. KNN-BERT: fine-tuning pre-trained models with KNN classifier. *arXiv preprint arXiv:2110.02523* 1, 2110 (2021), 8 pages.

[28] Joel Mackenzie, Kshitiz Gupta, Fang Qiao, Ahmed Hassan Awadallah, and Milad Shokouhi. 2019. Exploring user behavior in email re-finding tasks. In *The World Wide Web Conference*. Association for Computing Machinery, New York, NY, USA, 1245–1255.

[29] J MacQueen. 1967. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*. ACM, online, 281–297.

[30] Gloria Mark, Shamsi T Iqbal, Mary Czerwinski, Paul Johns, Akane Sano, and Yuliya Lutchyn. 2016. Email duration, batching and self-interruption: Patterns of email use on productivity and stress. In *Proceedings of the 2016 CHI conference on human factors in computing systems*. Association for Computing Machinery, New York, NY, USA, 1717–1728.

[31] Andrew McCallum, Kamal Nigam, et al. 1998. A comparison of event models for naive bayes text classification, In AAAI-98 workshop on learning for text categorization. *AAAI* 752, 41–48.

[32] Anders Miltner, Sumit Gulwani, Vu Le, Alan Leung, Arjun Radhakrishna, Gustavo Soares, Ashish Tiwari, and Abhishek Udupa. 2019. On the fly synthesis of edit suggestions. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–29.

[33] Stephen Muggleton and Luc de Raedt. 1994. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming* 19-20 (1994), 629–679. https://doi.org/10.1016/0743-1066(94)90035-3 Special Issue: Ten Years of Logic Programming.

[34] Ghulam Mujtaba, Liyana Shuib, Ram Gopal Raj, Nahdia Majeed, and Mohammed Ali Al-Garadi. 2017. Email classification research trends: review and open issues. *IEEE Access* 5 (2017), 9044–9064.

[35] Douglas Oard, William Webber, David A. Kirsch, and Sergey Golitsynskiy. 2015. Avocado Research Email Collection. https://catalog.ldc.upenn.edu/LDC2015T03.

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[37] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[38] Ronald L Rivest. 1987. Learning decision lists. *Machine learning* 2, 3 (1987), 229–246.

[39] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[40] Ahmad Sallab and Mohsen Rashwan. 2012. E-mail classification using deep networks. *Journal of Theoretical and Applied Information Technology JATIT*, 37 (03 2012).

[41] Xi'ao Su, Ran Wang, and Xinyu Dai. 2022. Contrastive Learning-Enhanced Nearest Neighbor Mechanism for Multi-Label Text Classification. In *Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Dublin, Ireland, 672–679.

[42] Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. 1997. Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning* 29, 1 (oct 1997), 5–44. https://doi.org/10.1023/A:1007413323501

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., Long Beach, US. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[44] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. 2001. Constrained K-Means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 577–584.

[45] Xiao-Lin Wang and Ian Cloete. 2005. Learning to classify email: a survey. In *2005 International Conference on Machine Learning and Cybernetics*. IEEE, Online, 5716 – 5719 Vol. 9. https://doi.org/10.1109/ICMLC.2005.1527956