An Adaptive Feature Selection Method for Learning-to-Enumerate Problem*

Satoshi Horikawa[1], Chiyonosuke Nemoto[2], Keishi Tajima[1][0000−0001−8226−3442], Masaki Matsubara[1][0000−0003−1950−683X], and Atsuyuki Morishima[1][0000−0003−4606−9065]

[1] University of Tsukuba, 1-2 Kasuga, Tsukuba, Ibaraki 305-8550 Japan
{satoshi.horikawa.2017b,chiyonosuke.nemoto.2014b}@mlab.info
{masaki,mori}@slis.tsukuba.ac.jp
[2] Kyoto University, Yoshida-Honmachi, Kyoto 606-8501 Japan
tajima@i.kyoto-u.ac.jp

Abstract. In this paper, we propose a method for quickly finding a given number of instances of a target class from a fixed data set. We assume that we have a noisy query consisting of both useful and useless features (e.g., keywords). Our method finds target instances and trains a classifier simultaneously in a greedy strategy: it selects an instance most likely to be of the target class, manually label it, and add it to the training set to retrain the classifier, which is used for selecting the next item. In order to quickly inactivate useless query features, our method compares discriminative power of features, and if a feature is inferior to any other feature, the weight 0 is assigned to the inferior one. The weight is 1 otherwise. The greedy strategy explained above has a problem of bias: the classifier is biased toward target instances found earlier, and deteriorate after running out of similar target instances. To avoid it, when we run out of items that have the superior features, we re-activate the inactivated inferior features. By this mechanism, our method adaptively shifts to new regions in the data space. Our experiment shows that our binary and adaptive feature weighting method outperforms existing methods.

Keywords: data extraction, ranking method, relevance feedback

1 Introduction

Suppose we want to quickly find a given number of instances of some target class from a fixed large data set. If we have enough sample data, we can train a classifier for the class, but sometimes we have no such training data. For example, when we want to find news articles related to some new topic from a large news corpus, we usually have no labeled training data for such a new topic.

In such a case, a possible strategy is to find target instances and train a classifier simultaneously. We select an item most likely to be of the target class

* This work was supported by JSPS KAKENHI Grant Number 22H00508, 23H03405, and JST CREST Grant Number JPMJCR22M2, Japan.

by using the current classifier (initially random), and manually label it. If it is of the target class, we add it to the set of found target instances. No matter what its label is, we also add it to the training data set, and re-train the classifier. We repeat this process until we obtain a given number of target instances. Our goal is to minimize the number of "misses", i.e., the number of non-target instances we manually label before obtaining a given number of target instances.

This problem is sometimes called *the learning-to-enumerate* problem [11], and is related to relevance feedback. Note that it is different from the active learning problem [23, 30, 13, 6, 1], where we want to obtain a good classifier with the minimum number of items we label regardless of their classes. In active learning, we choose an item to label that would best improve the classifier regardless of its class, but in our problem, we prefer to label instances of the target class.

In the learning-to-enumerate problem, therefore, there exists a trade-off between exploitation and exploration. We need to choose either an item that is more likely to be of the target class (exploitation), or an item that would better improve the classifier (exploration). Jörger et al. [11] conducted an experiment with 19 public data sets, and reported that an exploitation-only strategy with a random forest classifier achieved the best performance in most cases without any pattern regarding class bias, number of features, or total number of instances, that influenced their results in any consistent way.

One problem in the exploitation-only strategy for data extraction from a fixed data set is that the classifier is biased toward target instances found earlier. If the target instances are distributed across several clusters, the classifier is biased toward the clusters found earlier, and after we have extracted all the instances in those clusters, the performance of the classifier suddenly degrades.

Another issue in our problem setting is formulation of useful queries. Because we assume a new target class without existing training data, we do not know what features (e.g., keywords) we should use in the query. Therefore, we assume that we only have a noisy query including both useful and useless features.

We proposes *AdaFeaSE* (Adaptive Feature Selection for Enumeration), a method for data extraction from a fixed data set. It consists of the following two mechanisms for solving the two issues above. First, to quickly discard useless features, we compare discriminative power of features in a pairwise manner, and if a feature is inferior to any other with statistically significant difference, we inactivate it by giving weight 0. Otherwise a feature is active and given weight 1. Second, when some active feature runs out of matching items, we remove the feature from the set of the candidate features, re-activate inactive features, and compare their discriminative power again to re-select the features to use. By this mechanism, we can switch from an exhausted cluster to other clusters.

For example, suppose we want to find news articles on some topic from a news corpus by using keywords. There is, however, no single keyword that can find all the articles on the topic. Instead, the articles on the topic form several clusters, each of which corresponds to some keywords. We do not know those keywords in advance, and we can only come up with a set of keywords including both useful and useless ones. We run AdaFeaSE with these keywords as the features,

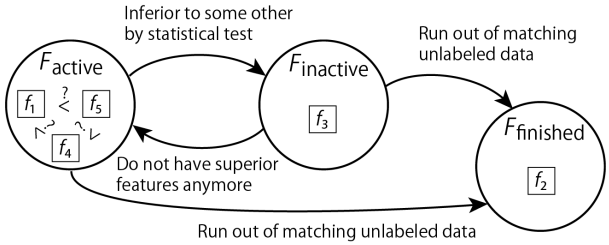


Fig. 1. State transition of features in our method. An active feature is inactivated if it is inferior to some other active feature by statistical test. On the contrary, an inactive feature is reactivated if no active feature is superior to it anymore. When an active or inactive feature runs out of matching unlabeled data, it is moved to the finished state.

and after several rounds of selecting and labeling items, suppose AdaFeaSE has found a useful keyword f that can find articles in one of the clusters with high precision. AdaFeaSE then inactivates the other keywords by assigning weight 0, and find target instances with relying on f. Because we extract articles from a fixed data set, we run out of articles including the keyword f at some point. AdaFeaSE then removes f, reactivate the other keywords that were given the weight 0, and compare their discriminative power to re-select the keywords to use. By this mechanism, AdaFeaSE focuses on each cluster in turn.

Fig. 1 summarizes the state transition of features among three states in AdaFeaSE: $\mathcal{F}_{active}$, $\mathcal{F}_{inactive}$, $\mathcal{F}_{finished}$. $\mathcal{F}_{finished}$ is the state representing that the feature has run out of the matching items.

We conducted experiments where we find articles related to given new topics from a fixed news corpus. To simulate a situation where we do not know what words are useful to find relevant articles, and have to come up with candidate words resulting in a set of words including both useful and useless ones, we collected candidate words through crowdsourcing. The results of our experiments show the superiority of our binary and adaptive feature selection method over the standard learning techniques.

2 Related Work

Feature selection has been an important research topic in machine learning [3, 9, 12, 2, 20, 15]. In particular, filter methods [21] use statistical tests such as Chi square tests for choosing prospective features. The filter methods test the correlation between features and the target class on the training data set, and either choose some fixed number of features with high correlation, or choose those with correlation higher than some threshold. On the other hand, AdaFeaSE tests whether each feature is significantly inferior to some other feature in the discriminative power, inactivate inferior ones temporarily, and reactivate them when the superior ones have run out of matching data items. By this mechanism, we adaptively change features to use. When the target class is distributed

over multiple clusters, useful features tend to be disjunctive, that is, each feature covers only one (or some) of the clusters. Our method chooses a small number of features effective for one (or some) of the clusters, and dynamically changes the features in order to focus on each cluster one by one.

Dimensionality reduction [26, 19, 27] is also a popular approach for reducing the number of features for improving the learning speed when we have many features. It is, however, effective when there are many redundant features that have strong correlation with each other. The problem we focus on is not a redundant feature set but a noisy feature set including many useless ones, e.g., a feature set that non-expert users come up with. Useless features in such a feature set do not necessarily have strong correlation with each other.

As explained before, we have a trade-off between exploitation and exploration in our problem setting. It is similar to the multi-armed bandit problem [16]. There have been studies that model feature selections as a kind of multi-armed bandit problem. Durand et al. [5] applied combinational bandits to online feature selection problem. In their setting, items arrive sequentially and the learner is allowed to access a small number of features for each item. Therefore, the problem is how to choose a subset of features to observe in order to maximize the classification rate. On the other hand, in our problem setting, we choose an item to label, and if we pay a constant cost for labeling it, we can observe all features of the item. In other words, in their setting, we can try one feature in one attempt, and we cannot choose an item to try, while in our setting, we can try all features in one attempt, and we need to choose one item to try.

As explained in the previous section, Jörger et al. [11] compared various methods of balancing exploitation and exploration in the learning-to-enumerate problem setting, and their conclusion is that a simple exploitation-only method is the best. This means that the methods for active learning, e.g., uncertainly sampling, which focus on exploration, do not perform well in our setting.

In our experiments, we collect candidate features (key phrases) through crowdsourcing, which results in a noisy feature set including many useless ones. There have been attempts to obtain features for classifications through crowdsourcing [4, 25, 32]. In those attempts, they show positive and negative samples to help workers to find useful features. By contrast, we assume that we do not have labeled sample data for training, thus cannot show such samples to workers.

A system proposed in [22] also uses human power to select useful features. When their system asks a user to label a document, it also picks words by using a standard feature selection method, and asks the user to determine if they are relevant to specific classes. This method can be combined with ours.

Binary weighting is not necessarily good for classification tasks [10, 17]. Our experimental result also show that our binary weighting method is inferior to some standard methods when they have been trained with enough data. However, our result also show that binary weighting can be a good quick approximation in the early stage of the training when we have a feature set including a few useful ones and many useless ones. In addition, our experimental result shows that feature sets human workers come up with really have such a characteristic.

Algorithm 1 The basic procedure of the exploitation-only strategy

Input: items $\mathcal{X}$, features $\mathcal{F}$, target class C, workers W, number of needed instances n
Output: set of found target instances $\mathcal{X}_{found}$

1: $\mathcal{X}_{unlabeled} \leftarrow \mathcal{X}$, $\mathcal{X}_{found} \leftarrow \emptyset$, $\mathcal{X}_{train} \leftarrow \emptyset$, initialize the model $\mathcal{M}$
2: **while** $|\mathcal{X}_{found}| < n$ **do**
3: **for all** $x_i \in \mathcal{X}_{unlabeled}$ **do** compute target class score p_i of x_i by using $\mathcal{M}$
4: $x \leftarrow x_i$ with the highest p_i
5: Query the workers W for the label y of x
6: **if** $y = C$ **then** add x to $\mathcal{X}_{found}$
7: Remove x from $\mathcal{X}_{unlabeled}$
8: Add (x, y) to $\mathcal{X}_{train}$
9: Re-train $\mathcal{M}$ on $\mathcal{X}_{train}$ for $\mathcal{X}_{unlabeled}$
10: **end while**

The exploitation-only strategy in the learning-to-enumerate problem is similar to relevance feedback. We query the labels of top-ranked items for improving rankings. Our work is particularly related to query reduction problem [8, 7, 14] where we remove words from verbose queries. Our method is different from theirs in that our method does not aim to find a set of features that best describes the whole target class. Instead, we find features with high precision even if they work only for a small subset of the target class, and we dynamically change the features to use in order to cover the whole target class.

3 Proposed Method

We first show the basic procedure of the exploitation-only strategy in the learning-to-enumerate problem in Algorithm 1. Let W be a set of human workers that always give the correct label to a queried item, $\mathcal{X}$ be the data set, and C be the target class. We repeatedly choose the most likely item $x \in \mathcal{X}$ and query W for its label y until we find n instances of C. To minimize the number of misses, i.e., labeling of non-target instances, we train a model $\mathcal{M}$, and choose the next item to label based on the target class score given by the current $\mathcal{M}$ (Lines 3-4). Every time we label a new item, we re-train the model $\mathcal{M}$ (Lines 7-9).

This procedure always exploits the current knowledge, and never explore new regions in the data space. It has been reported in [11] that the exploitation-only strategy is the best for the learning-to-enumerate problem. Based on their results, we adopt the exploitation-only strategy as the basis of our method.

In our framework, human workers play two roles. First, we ask workers W' to propose prospective features $\mathcal{F}$. We also ask workers W to label chosen items at Line 5 in Algorithm 1. We assume that labels given by W (by using majority voting) are always correct, while features proposed by W' are noisy because suggesting a candidate feature is a difficult open question, while determining whether an item belong to a specific class is a closed question.

Next we explain our method AdaFeaSE. Let $\mathcal{F} = \{f_1, \ldots, f_m\}$ be the set of features proposed by the workers W'. Let $f_j(x_i)$ denote the value of the feature

f_j of the item x_i. In AdaFeaSE, $\mathcal{M}$ is represented by $(\mathcal{F}_{active}, \mathcal{F}_{inactive}, \mathcal{F}_{finished})$, where the three components are the following subsets of $\mathcal{F}$:

- $\mathcal{F}_{active}$: currently active features,
- $\mathcal{F}_{inactive}$: currently inactive features,
- $\mathcal{F}_{finished}$: features that no remaining unlabeled item has.

They are disjoint decomposition of $\mathcal{F}$, that is:
$\mathcal{F}_{active} \cap \mathcal{F}_{inactive} = \mathcal{F}_{inactive} \cap \mathcal{F}_{finished} = \mathcal{F}_{active} \cap \mathcal{F}_{finished} = \emptyset$, and
$\mathcal{F}_{active} \cup \mathcal{F}_{inactive} \cup \mathcal{F}_{finished} = \mathcal{F}$.

When we initialize $\mathcal{M}$, all features in $\mathcal{F}$ are in $\mathcal{F}_{active}$. When updating $\mathcal{M}$, if no remaining unlabeled item has f_i (when the feature is not binary, we use a threshold), we move f_i to $\mathcal{F}_{finished}$. In addition, if we determine that $f_i \in \mathcal{F}_{active}$ is inferior to some $f_j \in \mathcal{F}_{active}$, we move f_i to $\mathcal{F}_{inactive}$. On the contrary, if no $f_j \in \mathcal{F}_{active}$ is superior to $f_i \in \mathcal{F}_{inactive}$ anymore (because of the update of the statistics or state transition of f_j to $\mathcal{F}_{finished}$), we move f_i back to $\mathcal{F}_{active}$.

By this initialize and update procedure, each feature moves among three states in the following way. First, all features are in the active state. Each feature then may go back and forth between the active and inactive state. However, once it moves to the finished sate, it never moves back to the other states (Fig. 1).

In order to compare features f_i and f_j in terms of the discriminating power to the target class, we maintain a table $\mathcal{S}$ where i-th (and j-th) row stores the number of target and non-target instances in $\mathcal{X}_{train}$ that has the feature f_i (and f_j). To determine whether one is inferior to the other with statistically significant difference, we use Fischer's exact test when at least one of the four parameters in the contingency table is smaller than 5, and we use Chi-squared test otherwise.

Algorithm 2 shows the sub-routines used by AdaFeaSE for initializing and updating $\mathcal{M}$, and for computing the target score p_i for x_i at the corresponding parts in Algorithm 1. In the procedure for re-training $\mathcal{M}$ in Algorithm 2, we first update $\mathcal{S}$ based on the current $\mathcal{X}_{train}$. After that we first revoke features in $\mathcal{F}_{active}$ and $\mathcal{F}_{inactive}$ that no remaining data items in $\mathcal{X}_{unlabeled}$ has (Line 7-8). Next, we reactivate all f in $\mathcal{F}_{inactive}$ that have no active superior f' anymore (Line 9-11). We compare f and f' based on the current $\mathcal{S}$. Some f may be reactivated because of the updated $\mathcal{S}$, and some f may be reactivated because some f' that was superior to f has moved to $\mathcal{F}_{finished}$. Finally, we temporarily inactivate all f that are inferior to any f' that is currently active (Line 12-14).

$\mathcal{M}$ computes the target class score p_i of x_i by the formula $p_i = \sum_{f \in \mathcal{F}_{active}} f(x_i)$. In other words, we compute p_i by a linear combination of the features with the binary weight 1 for active features and 0 for other features. We use p_i for ranking the remaining data items based on their likeliness of being of the target class.

The theorem below shows the time complexity of AdaFeaSE.

Theorem 1. *The worst case time complexity of AdaFeaSE is* $O(|\mathcal{F}|^2|\mathcal{X}|)$. □

It is proportional to $|\mathcal{F}|^2$ in the worst case because of the pairwise comparison of the features. However, the worst case occurs only when $O(|\mathcal{F}|)$ rows of the table $\mathcal{S}$ are updated (thus we need to re-evaluate all pairs of them), which rarely

Algorithm 2 AdaFeaSE (sub-routines invoked from Algorithm 1)

procedure Initialize $\mathcal{M}$
  $\mathcal{F}_{active} \leftarrow \mathcal{F},\ \mathcal{F}_{inactive} \leftarrow \emptyset,\ \mathcal{F}_{finished} \leftarrow \emptyset$
end procedure

procedure Re-train $\mathcal{M}$ on $\mathcal{X}_{train}$ for $\mathcal{X}_{unlabeled}$
  Update the statistics table $\mathcal{S}$ based on $\mathcal{X}_{train}$
  for all $f \in \mathcal{F}_{active} \cup \mathcal{F}_{inactive}$ **do**
    if no $x \in \mathcal{X}_{unlabeled}$ has the feature f **then** move f to $\mathcal{F}_{finished}$
  for all $f \in \mathcal{F}_{inactive}$ **do**
    if no $f' \in \mathcal{F}_{active}$ is superior to f statistically significantly based on $\mathcal{S}$ **then**
      move f to $\mathcal{F}_{active}$
  for all $f \in \mathcal{F}_{active}$ **do**
    if f is inferior to any $f' \in \mathcal{F}_{active}$ statistically significantly based on $\mathcal{S}$ **then**
      move f to $\mathcal{F}_{inactive}$
end procedure

procedure Compute target class score p_i of x_i by $\mathcal{M}$
  $p_i \leftarrow \sum_{f \in \mathcal{F}_{active}} f(x_i)$
end procedure

happens in practice. The results of our experiments with the data from a real application show that only a few rows are updated in the most cases. Thus we expect that AdaFeaSE scales well in practice with the number of features in $\mathcal{F}$.

4 Experiments

We conducted experiments to evaluate our method. We used 6,684 news articles published by Yahoo! Japan [29] in 2016. We set two target classes. Class Scandal is the class of articles on scandals of celebrities. We define celebrities as people who sometimes appear on TV as their jobs. Class Toyota is the class of articles on Toyota's business performance or on events that can affect it.

To create the ground truth, we crowdsourced the tasks of giving two binary labels corresponding to the two classes to each article through Yahoo! Crowdsourcing [28]. We assigned three workers to each article, and adopted their results if they all agree on the label. If they do not agree, we hired another three workers to give labels, and adopted the results of majority voting. Table 1 summarizes the results. Only a small portion of the 6,684 articles belong to the target classes.

We also submitted 200 microtasks to Yahoo! Crowdsourcing to collect phrases that workers think are good clues for distinguishing articles of each target class. We obtained 386 and 656 phrases for the Scandal and Toyota classes, respectively. For example, the phrases for Toyota class include: automatic driving and crude oil. We regard each obtained phrase as a binary feature of the articles; it has the value 1 if the phrase appears in the article and 0 otherwise.

The phrases nominated by crowds include many phrases that appear in no article in our data set. Out of 386 and 656 phrases nominated for Scandal and

Table 1. Statistics of data sets

	Articles			Phrases	
Target Class	Positive	Negative	Total	Nominated	Used
Scandal	252	6,432	6,684	386	286
Toyota	79	6,605	6,684	656	272

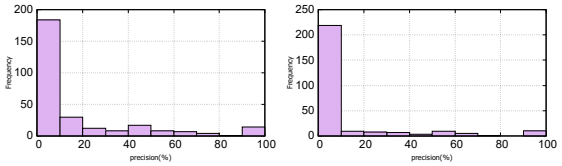


Fig. 2. Histogram over the precision of features (left: Scandal, right: Toyota).

Toyota class, 286 and 272 phrases, respectively, appear in at least one article, as summarized in Table 1. The other phrases were excluded from our experiment.

Fig. 2 shows the histograms of the number of phrases nominated for Scandal (left) and Toyota (right) distributed over the precision, where the precision represents the ratio of the number of true positive articles to the number of articles including the phrase. As shown in these histograms, most phrases have very low precision while there are also some phrases with high precision.

Those high-precision phrases, however, include phrases that appear only in a couple of articles. In our adaptive feature selection method, a high-precision feature can be useful even if its recall is low, but a feature that only a couple of instances have may not be useful because we may run out of the data with the feature before the classifier learns that the feature is useful. Because of that, there are a very small number of features that are really useful in our data sets.

Even if the precision is low, if a feature has strong negative correlation, it is useful in discriminating target instances. Fig. 3 shows the pointwise mutual information $\log_2 (P(q|p)/P(q))$ between p meaning that the phrase appears in the article, and q meaning that the article is the target instance. It shows that few features have negative mutual information, and they only have small quantity of information compared with those with positive one. Similar results are expected when features are nominated by human users. Our binary weighting method cannot give negative weights to features while the standard learning methods can, but it is not a significant disadvantage of our method when we use features nominated by human users because of these properties of such features. When we allow users to nominate a feature as a negative feature, e.g., a keyword whose absence suggests that the document is a target instance, we can flip its value, i.e., we define the feature takes the value 1 when the instance does not have it.

4.1 Experimental Result

We ran our method and six baseline methods on our data set to compare their performance. We ran each method for two tasks, one for Scandal class and one

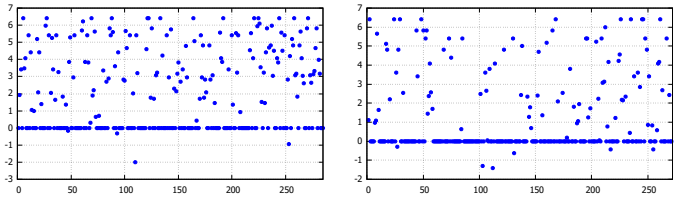


Fig. 3. Pointwise mutual information of features and class (left: Scandal, right: Toyota)

Table 2. Precision@k of AdaFeaSE with different α values

	Scandal			Toyota		
Method (α)	@100	@500	@1K	@100	@500	@1K
AdaFeaSE (0.05)	.468	.251	.179	.219	.079	.055
AdaFeaSE (0.10)	.448	.275	.180	.213	.073	.056
AdaFeaSE (0.15)	.444	.257	.180	.226	.076	.056

for Toyota class. The six baselines include logistic regression (LR), random forest (RF), which [11] concluded is the best in the learning-to-enumerate problem, and Lasso. We also included the combination of PCA (Principal Component Analysis) and these three baseline methods. We first run PCA (cumulative proportion threshold is 0.8) on the data set with the nominated features to obtain the data set in the reduced dimension, and run one of the baselines on that data set.

Note that our goal is to quickly find target instances when starting with no training data set, and the neural network-based methods [6, 18, 24, 31], which require a large training data set, are not effective in our problem setting.

In Algorithm 1, we ask crowds to label a data (Line 5). In this experiment, we assume that they always return the correct answers, i.e., the ground truth we created by hiring three or six workers as explained before. We leave an experiment with crowd answers that are not always correct for future work.

AdaFeaSE has a parameter α, which is the level of significance for statistical test. First, we run AdaFeaSE with three α values: $\alpha = 0.05, 0.10, 0.15$. Table 2 shows Precision@k of the three methods for $k = 100, 500, 1000$ for the two tasks. Presision@k here means that the ratio of the true positives in the k data items chosen by the first k rounds of the while loop in Algorithm 1. Table 2 shows that these α values do not largely affect the performance of AdaFeaSE. In other words, AdaFeaSE is very robust and is not sensitive to the parameter α.

Fig. 4 shows the recall of AdaFeaSE and the baselines after each round in the two tasks (left: Scandal, right: Toyota). Because these methods include randomness (they randomly choose an instance when there are ties), we run each method five times and take the average of the recall values. For AdaFeaSE, we chose the best α in each case. Because our goal is to quickly find some number of target instances, the performance at the early rounds is of our interest. Therefore, Fig. 4 only shows the early rounds upto the 500th round. These graphs show that AdaFeaSE outperforms the baselines at those early rounds.

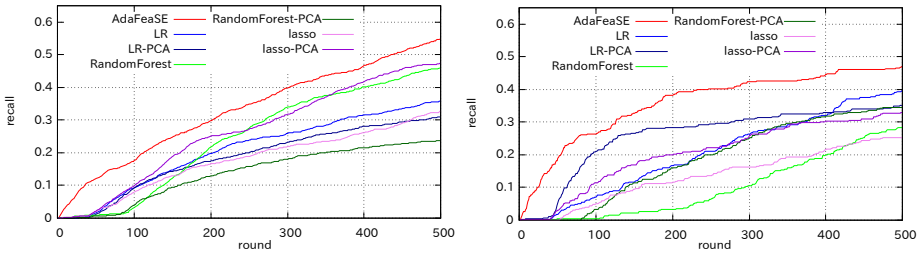


Fig. 4. Recall of AdaFeaSE and baselines at early rounds (left: Scandal, right: Toyota)

Table 3. Performance of AdaFeaSE and the second best methods (@50, @100, @300)

Class	Round	Recall (AdaFeaSE)	Second Best	Recall (Second)	Diff	Ratio
Scandal	@50	0.119	Lasso + PCA	0.021	0.098	567%
	@100	0.178	Lasso + PCA	0.099	0.079	180%
	@300	0.398	RF	0.340	0.058	117%
Toyota	@50	0.172	LR + PCA	0.053	0.119	324%
	@100	0.273	LR + PCA	0.210	0.063	130%
	@300	0.425	LR + PCA	0.309	0.116	138%

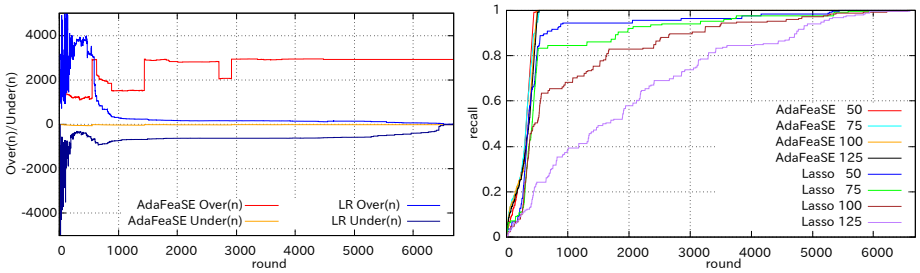
Table 3 shows the comparison of AdaFeaSE and the second best methods in the early rounds. It shows that the difference is significant. AdaFeaSE is outperformed by some others (not necessarily the second best one in Table 3) when they are trained enough. However, when we want to quickly find a small number of target instances, such as 30 instances, AdaFeaSE can find them earlier.

4.2 Analysis on why AdaFeaSE works

In our experiment, AdaFeaSE outperformed the baselines in the earlier rounds. The next question is why it works. We made the following hypotheses: First, by inactivating features that are inferior to any others, AdaFeaSE can avoid overestimation of useless features at the early rounds. Second, because AdaFeaSE detects features running out of matching items, and adaptively changes the features to use, it works better when the target class is distributed across many clusters. We conducted experiments for validating these two hypotheses.

To validate the first hypothesis, we measured how over/under-estimation of the weights change as we proceed through the rounds. We choose logistic regression for the comparison, and use the Scandal data set. We do not choose methods with PCA for the comparison because their feature sets are different from that of AdaFeaSE, and we cannot directly compare over/under-estimation of the features. We choose logistic regression because it was the method that retrieved all the target instances at the earliest round, and we use its final weights as the reference value for defining over/under-estimation as we explain below.

We first define two measures for evaluating the ratio of overestimation and underestimation. Let $W_t = \langle w_{t,1}, \ldots, w_{t,m} \rangle$ be the vector of weights for the



(a) (b)

Fig. 5. (a) Over/under-estimation of weights at each round, (b) Recall by AdaFeaSE and Lasso at each round for generated data sets with different numbers of clusters.

features $f_1, \ldots, f_m$ at round t. Let also $W = \langle w_1, \ldots, w_m \rangle$ be the final weights output by logistic regression for the features $f_1, \ldots, f_m$ after all 6,684 articles are added to the training set. Because logistic regression could retrieve all the target instances earlier than AdaFeaSE, we regard the final weights output by logistic regression as the best available approximation of the ideal weights.

We then normalize the vector W_t by dividing the coefficients by $\sum_j |w_{t,j}|$. Let $\bar{W}_t = \langle \bar{w}_{t,1}, \ldots, \bar{w}_{t,m} \rangle$ denote the normalized vector. That is, $\bar{w}_{t,i} = w_{t,i} / \sum_j |w_{t,j}|$. The normalized final weights, $\bar{w}_i$, are similarly defined by $\bar{w}_i = w_i / \sum_j |w_j|$.

We then define the overestimation ratio and underestimation ratio of weights at the round t as follows:

$$Over(t) = \sum_{i=1}^{m} \begin{cases} \frac{\bar{w}_{t,i} - \bar{w}_i}{|\bar{w}_i|} & (\bar{w}_{t,i} > \bar{w}_i) \\ 0 & (otherwise) \end{cases} \tag{1}$$

$$Under(t) = \sum_{i=1}^{m} \begin{cases} \frac{\bar{w}_{t,i} - \bar{w}_i}{|\bar{w}_i|} & (\bar{w}_{t,i} < \bar{w}_i) \\ 0 & (otherwise) \end{cases} \tag{2}$$

They represent how feature weights at round t deviate from the final weights output by logistic regression. Fig. 5 (a) shows $Over(t)$ and $Under(t)$ for AdaFeaSE (red line) and logistic regression (blue line). The x-axis is the round $t = 1$ to $6,684$, and y-axis is $Over(t)/Under(t)$. Values closer to 0 is better. Fig. 5 (a) shows that AdaFeaSE has lower $Over(t)$ in the early rounds. We believe this is one of the reasons why AdaFeaSE could find more target instances at the early rounds. On the other hand, logistic regression has lower $Over(t)$ at $t > 700$. $Over(t)$ of AdaFeaSE largely increases at several t where some useful features are moved to $\mathcal{F}_{finished}$, and less useful features are re-activated.

Next, to validate the second hypothesis, we conducted an experiment with generated data sets consisting of different number of clusters. We generated four data sets, those with 50, 75, 100, and 125 clusters. Each of them consists of 6,684 documents, among which 252 are target instances. We also assume that 250 phrases are used as the features. These numbers are the same as those of our Scandal data set. The four data sets are different from each other only in the

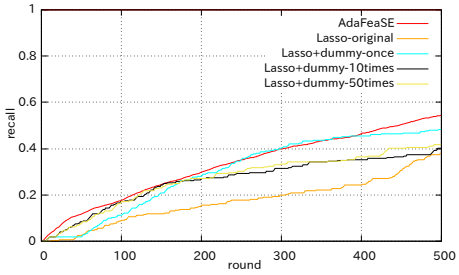


Fig. 6. Recall by pre-trained Lasso simulating initial weight values 1 and AdaFeaSE.

number of features that only target instances have. Each data set includes 50, 75, 100, and 125 phrases that appear only in the target instances, respectively. We call these features positive features. Because they appear only in the target instances, precision of these features is 1. They are also disjoint with each other, i.e., no document includes more than one positive feature. They are also complete, i.e., every target instance has one of the positive features. Therefore, the recall totally achieved by all the positive features is 1. Because of this assignment of features, each positive feature forms a cluster. Therefore, each data set includes 50, 75, 100, and 125 clusters, respectively. Because we have 250 features, each data set has 200, 175, 150, and 125 features that are not positive features, respectively. Precision of these features are set to 0.04, which is the precision of features that randomly appear in both target and non-target instances.

Fig. 5 (b) shows the performance of AdaFeaSE and Lasso on these four data sets. We chose Lasso for this comparison because it was the second best method at the early rounds for the scandal data set. The performance of Lasso deteriorates as the number of clusters increases. On the other hand, the performance of AdaFeaSE does not largely change when the number of clusters increases. This result supports our hypothesis that one of the reason of the superiority of AdaFeaSE is its ability to switch from a cluster to a cluster.

4.3 Lasso with Initial Weight Values Being 1

One difference between AdaFeaSE and the baselines is the initial weight of the features. In AdaFeaSE, all features are initially active and given weight 1. In the other methods, the initial weights of features are 0. Because we use features that are nominated by human workers as prospective, this difference may affect the performance of AdaFeaSE and the other methods at the very early rounds.

To examine if it is one of the reasons of the difference of the performance, we conducted another experiment simulating Lasso with the initial weights 1. In order to simulate it, we pre-train Lasso with 1, 10, or 50 positive samples that has all the features, and use this pre-trained classifier as the initial classifier.

Fig. 6 shows the result. This result shows that this pre-training improves the performance of Lasso at the very early rounds, but it is still outperformed by AdaFeaSE up to around the 240th round.

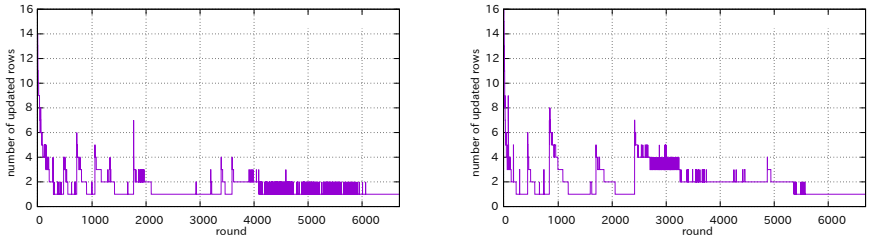


Fig. 7. The number of updated rows in $\mathcal{S}$ at each round (left: Scandal, right: Toyota).

4.4 Scalability of the Algorithm

As explained before, the time complexity of AdaFeaSE is $O(|\mathcal{F}|^2|\mathcal{X}|)$ in the worst case, where $O(|\mathcal{F}|)$ rows in the statistics table $\mathcal{S}$ are updated in every round. We measured how many rows were actually updated in our experiments. Fig. 7 shows the numbers of updated rows in each round for Scandal (with $|\mathcal{F}| = 386$) and Toyota (with $|\mathcal{F}| = 656$) data sets. As shown in these graphs, except for the very beginning, less than 10 rows are updated. In addition, the comparison of Scandal case and Toyota case suggests that the numbers of updates do not increase proportionally to $|\mathcal{F}|$. Therefore, we conclude that AdaFeaSE scales well with the number of nominated features. In fact, the computation time of AdaFeaSE is far shorter than the other baseline methods in our experiment.

5 Conclusions

In this paper, we focus on the situations where we want to quickly find some number (not all) of items of the target class from a fixed data set, but we only have very noisy feature sets including some useful ones and many useless ones.

Noisy feature sets usually require more training data, and it conflicts with our goal of quickly finding some data. We proposed a learning method that can quickly discard useless features. We compare features in a pairwise way based on their discriminative power, and if a feature is inferior to any other feature with a statistically significant difference, we temporarily inactivate the inferior one.

Another issue in our problem setting is that the classifier is biased toward positive instances found earlier. To avoid that, when we run out of remaining items that have the superior features, we re-evaluate inactivated features. By this strategy, even when positive instances are distributed across several clusters, our method can focus on those clusters one by one.

We compared our method with several baselines using two data sets from a real application and found that our method is superior to them in the performance at the early rounds. When we want to quickly find 30 or 50 instances of the target class, our method can find them earlier than the baselines.

An interesting remaining issue is how to switch from our method, which is good at early rounds, to other methods, which are good at later rounds. Such a method would enable us to apply our method to a wider range of applications.

References

1. Beluch, W.H., Genewein, T., Nürnberger, A., Köhler, J.M.: The power of ensembles for active learning in image classification. In: Proc. of CVPR (2018)
2. Cai, J., Luo, J., Wang, S., Yang, S.: Feature selection in machine learning: A new perspective. Neurocomputing **300**, 70–79 (2018)
3. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers & Electrical Engineering **40**(1), 16–28 (2014)
4. Cheng, J., Bernstein, M.S.: Flock: Hybrid crowd-machine learning classifiers. In: Proc. of CSCW. pp. 600–611 (2015)
5. Durand, A., Gagné, C.: Thompson sampling for combinatorial bandits and its application to online feature selection. In: Proc. of AAAI Conference Workshop on Sequential Decision-Making with Big Data. pp. 6–9 (2014)
6. Gal, Y., Islam, R., Ghahramani, Z.: Deep Bayesian active learning with image data. In: Proc. of ICML. pp. 1183–1192. PMLR (2017)
7. Ganguly, D., Leveling, J., Magdy, W., Jones, G.J.: Patent query reduction using pseudo relevance feedback. In: Pro. of CIKM. pp. 1953–1956 (2011)
8. Gupta, M., Bendersky, M.: Information retrieval with verbose queries. In: Proc. of SIGIR. pp. 1121–1124 (2015)
9. Hall, M.A., Smith, L.A.: Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper. In: Proc. of Intl. Florida Artificial Intelligence Research Society Conf. pp. 235–239. AAAI Press (1999)
10. Huang, P., Bu, J., Chen, C., Qiu, G.: An effective feature-weighting model for question classification. In: Proc. of Intl. Conf. on Computational Intelligence and Security. pp. 32–36 (2007)
11. Jörger, P., Baba, Y., Kashima, H.: Learning to enumerate. In: Intl. Conf. on Artificial Neural Networks. pp. 453–460. Springer (2016)
12. Khalid, S., Khalil, T., Nasreen, S.: A survey of feature selection and feature extraction techniques in machine learning. Proc. of Science and Information Conf. pp. 372–378 (2014)
13. Konyushkova, K., Sznitman, R., Fua, P.: Learning active learning from data. arXiv preprint arXiv:1703.03365 (2017)
14. Koopman, B., Cripwell, L., Zuccon, G.: Generating clinical queries from patient narratives: A comparison between machines and humans. In: Proc. of SIGIR. pp. 853–856 (2017)
15. Kou, G., Yang, P., Peng, Y., Xiao, F., Chen, Y., Alsaadi, F.E.: Evaluation of feature selection methods for text classification with small datasets using multiple criteria decision-making methods. Applied Soft Computing **86**, 105836 (2020)
16. Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics **6**(1), 4–22 (1985)
17. Lan, M., Tan, C.L., Low, H.: Proposing a new term weighting scheme for text categorization. In: Proc. of National Conf. on Artificial Intelligence. pp. 763–768 (2006)
18. Qi, Y., Zhang, J., Liu, Y., Xu, W., Guo, J.: Cgtr: Convolution graph topology representation for document ranking. In: Proc. of CIKM. p. 2173–2176 (2020)
19. Reddy, G.T., Reddy, M.P.K., Lakshmanna, K., Kaluri, R., Rajput, D.S., Srivastava, G., Baker, T.: Analysis of dimensionality reduction techniques on big data. IEEE Access **8**, 54776–54788 (2020)
20. Remeseiro, B., Bolon-Canedo, V.: A review of feature selection methods in medical applications. Computers in Biology and Medicine **112**, 103375 (2019)

21. Sánchez-Maroño, N., Alonso-Betanzos, A., Tombilla-Sanromán, M.: Filter methods for feature selection: A comparative study. In: Proc. of IDEAL. pp. 178–187 (2007)
22. Settles, B.: Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In: Proc. of EMNLP. pp. 1467–1478. ACL (2011)
23. Settles, B.: Active learning. Synthesis lectures on artificial intelligence and machine learning **6**(1), 1–114 (2012)
24. Sun, X., Tang, H., Zhang, F., Cui, Y., Jin, B., Wang, Z.: Table: A task-adaptive bert-based listwise ranking model for document retrieval. In: Proc. of CIKM. p. 2233–2236 (2020)
25. Takahama, R., Baba, Y., Shimizu, N., Fujita, S., Kashima, H.: Adaflock: Adaptive feature discovery for human-in-the-loop predictive modeling. In: Proc. of AAAI Conf. pp. 1619–1626 (2018)
26. Van Der Maaten, L., Postma, E., Van den Herik, J.: Dimensionality reduction: a comparative. Journal of Machine Learning Research **10**(66-71), 13 (2009)
27. Weinberger, K.Q., Sha, F., Saul, L.K.: Learning a kernel matrix for nonlinear dimensionality reduction. In: Proc. of ICML. p. 106 (2004)
28. Yahoo! crowdsourcing, http://crowdsourcing.yahoo.co.jp/
29. Yahoo! news, http://news.yahoo.co.jp/
30. Yu, H., Yang, X., Zheng, S., Sun, C.: Active learning from imbalanced data: A solution of online weighted extreme learning machine. IEEE Trans. on Neural Netw. Learn. Syst. **30**(4), 1088–1103 (2018)
31. Zhang, J., Geng, Y.a.o., Li, Q., Shi, C.: More than one: A cluster-prototype matching framework for zero-shot learning. In: Proc. of CIKM. p. 1803–1812 (2020)
32. Zou, J.Y., Chaudhuri, K., Kalai, A.T.: Crowdsourcing feature discovery via adaptively chosen comparisons. In: Proc. of AAAI HComp. pp. 198–205 (2015)