AdaFlock: Adaptive Feature Discovery for Human-in-the-Loop Predictive Modeling

Ryusuke Takahama
scouty Inc.*
r.takahama7591@gmail.com

Yukino Baba
Kyoto University
baba@i.kyoto-u.ac.jp

Nobuyuki Shimizu, Sumio Fujita
Yahoo Japan Corporation
{nobushim, sufujita}@yahoo-corp.jp

Hisashi Kashima
Kyoto University; RIKEN Center for AIP
kashima@i.kyoto-u.ac.jp

Abstract
Feature engineering is the key to successful application of machine learning algorithms to real-world data. The discovery of informative features often requires domain knowledge or human inspiration, and data scientists expend a certain amount of effort into exploring feature spaces. Crowdsourcing is considered a promising approach for allowing many people to be involved in feature engineering; however, there is a demand for a sophisticated strategy that enables us to acquire good features at a reasonable crowdsourcing cost. In this paper, we present a novel algorithm called AdaFlock to efficiently obtain informative features through crowdsourcing. AdaFlock is inspired by AdaBoost, which iteratively trains classifiers by increasing the weights of samples misclassified by previous classifiers. AdaFlock iteratively generates informative features; at each iteration of AdaFlock, crowdsourcing workers are shown samples selected according to the classification errors of the current classifiers and are asked to generate new features that are helpful for correctly classifying the given examples. The results of our experiments conducted using real datasets indicate that AdaFlock successfully discovers informative features with fewer iterations and achieves high classification accuracy.

1 Introduction
Feature engineering is the key to the successful design of machine learning classifiers for real-world data. The learning of a good predictive model is difficult if we fail to determine useful features for classification. The discovery of informative features often requires domain knowledge or human inspiration. Considerable efforts have been made in machine learning projects to explore feature spaces. Researchers in several fields have attempted to construct task-specific features by incorporating heuristics. For instance, in the computer vision area, SIFT (Lowe 2004) and SURF (Bay, Tuytelaars, and Gool 2006) were designed for general object recognition tasks, and different features were proposed for detailed tasks such as face detection (Viola and Jones 2004) or human body detection (Dalal and Triggs 2005).

Although careful feature engineering for a particular task would be beneficial for improving classification performance, the broad exploration of features is quite laborious. The recent expansion of crowdsourcing platforms allows many people to be involved in feature engineering. Although it is difficult for non-expert crowdsourcing workers to provide correct answers for fine-grained classification tasks, they may have capabilities to determine informative features for classification (Branson et al. 2010).

Recently, a few methods for crowdsourced feature discovery have been proposed. Cheng and Bernstein developed an interactive platform called Flock (Cheng and Bernstein 2015), which asks crowdsourcing workers to compare paired examples and describe their differences. The collected descriptions are organized as features, such as “does this painting contain flowers?”; feature labels (i.e., “yes” or “no”) for each example are then generated using another set of crowdsourcing tasks. The paired examples are randomly chosen by Flock; however, features may be able to be generated in a more efficient manner by adaptively selecting the exam-
plies. Zou et al. proposed an adaptive method for selecting triples of examples to avoid the generation of overlapping features (Zou, Chaudhuri, and Kalai 2015). This method chooses examples based on feature labels of the previously discovered features. Because the method does not focus on supervised learning, the performance of a classifier is not considered during example selection.

In this paper, we propose a new algorithm called AdaFlock to efficiently generate informative features through crowdsourcing. Our algorithm aims to obtain features helpful for improving the classification performance through iterations. AdaFlock is inspired by AdaBoost (Freund and Schapire 1997), which iteratively trains weak classifiers by increasing the weights of examples misclassified by the current classifiers. Analogously, at each iteration of AdaFlock, crowdsourcing workers are shown examples selected according to the classification errors of the current classifiers (Figure 1(a)). The workers are asked to generate features helpful for correctly classifying the given examples (Figure 1(b)). AdaFlock then asks crowdsourcing workers to label each example based on each feature definition (Figure 1(c)). A weak classifier is trained by using the obtained labels, and the Filter function (Bradley and Schapire 2007) is applied for resampling examples according to the classification errors of the current classifier (Figure 1(d)). The sampled examples are shown to workers of the next iteration.

Table 1 summarizes the difference between AdaFlock and the other crowdsourced feature discovery methods. Although the method proposed by Branson et al. (2010) requires predefined feature definitions, AdaFlock uses crowdsourcing for defining features. In contrast to Flock (Cheng and Bernstein 2015), AdaFlock adaptively generates features to improve the classification accuracy. While adaptive triple selection (Zou, Chaudhuri, and Kalai 2015) aims to obtain diverse features, the goal of AdaFlock is to obtain informative features to improve the classification accuracy.

We conducted experiments on a crowdsourcing platform by using image and movie classification datasets and observed that AdaFlock discovers various types of features not covered by Flock. Moreover, a classifier built by using AdaFlock outperforms the classifier built through Flock.

It would be worth noticing that one of the major drawbacks of crowdsourced feature discovery is scalability; this approach is not suitable for classifying a large number of examples because of the requirement of crowdsourced feature labeling. Practical situations where crowdsourced feature discovery is useful are (1) obtained interpretable features themselves are important for explaining predictions, and (2) it is desired to create an accurate classifier regardless of crowdsourcing costs.

The contributions of this paper are threefold:

- We address the problem of adaptive crowdsourced feature generation for supervised learning.
- We propose a novel algorithm called AdaFlock, which obtains informative features by sampling difficult examples for the current classifier.
- Through experiments conducted using actual crowdsourcing datasets, we confirm that AdaFlock achieves better classification performance than the existing methods.

2 AdaFlock

2.1 Problem Setting

We first show a formulation of our feature generation problem for predictive modeling. We focus on binary classification in this paper. Assume that there is a training dataset as $D = \{ (x_i, y_i) \}_{i=1}^N$, where $y_i \in \{-1, +1\}$. Given the training dataset, our goal is to build a classifier $H (x)$. Unlike typical binary classification problems, a feature vector of each $x_i$ is not given here. Thus, we will concurrently generate feature vectors and train a classifier.

2.2 Overview

We propose a novel algorithm called AdaFlock to efficiently generate feature vectors through crowdsourcing and to train a classifier by using the obtained features. AdaFlock requests crowdsourcing workers to process feature-definition tasks and feature-labeling tasks. Examples of these two tasks are illustrated in Figure 3 and 4. In feature-definition tasks, workers are shown a small number of positive and negative examples and asked to describe the difference between the positive and negative examples. This approach is called analogical encoding, and its advantages for producing predictive features have been demonstrated in (Cheng and Bernstein 2015). AdaFlock instructs workers to write a description as a yes–no question; for example, “is the sky illustrated in this painting?” or “does this text contain any hard number?” In feature-labeling tasks, workers are given.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crowdsourced feature-definition</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Crowdsourced feature-labeling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Adaptive example selection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Supervised learning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Figure 2: Overview of AdaFlock algorithm. At the $t$-th iteration, given the weights of examples, AdaFlock first resamples a small number of positive and negative examples, $S_F^+ \subseteq D^+$ and $S_F^- \subseteq D^-$. AdaFlock shows $S_F^+$ and $S_F^-$ to crowdsourcing workers and requests them to process feature-definition tasks to obtain $F_t$. Feature-labeling tasks are then posted, in which workers are asked to label each object $x_i$ based on each feature definition $f_j \in F_t$. AdaFlock next trains a weak leaner by using the obtained labels, and update the current leaner and the weight of each example, $w_{t+1}^i$.

Figure 3: Example of a feature-definition task, where workers are shown a small number of positive and negative examples and asked to describe the difference between them. Workers are instructed to write a description as a yes-no question.

Figure 4: Example of a feature-labeling task, in which workers are given an example and asked to provide a binary label for each question collected in the feature-definition tasks.

In step (a), AdaFlock uses crowdsourcing to obtain new features based on the given data distribution, and trains a classifier by using the obtained features. The data distribution is derived from the weights of the examples according to the classification errors in the previous iteration. The crowdsourcing workers are required to define features helpful for correctly classifying the misclassified examples in the previous iteration. In other words, we intend for the workers to consider data distribution when they define features; however, data distribution is less interpretable for humans. We follow the solution proposed by Pareek and Ravikumar (2013) and apply the filtering approach to resample a small number of examples. The sampled examples are shown to the workers.

In step (b), AdaFlock computes the weighted accuracy of the weak learner and incorporates the learner into the current classifier. The weights are calculated based on the prediction accuracies for all the training samples. In the step (c), the data distribution is updated so that the examples misclassified by the current classifier are more likely to be sampled in step (a) of the next iteration. AdaFlock repeats these steps.
Algorithm 1 AdaFlock

1: input Crowd oracle \( O \); training dataset \( D \);
   termination parameters \( \delta_t \) and \( \epsilon \);
   number of examples for each iteration \( m \);
   number of feature definitions at each iteration \( k \);
   number of iterations \( T \)
2: Set weights \( w_1^t = 1/N, i = 1, \ldots, N \)
3: \( D^+ = \{(x,y) \mid (x,y) \in D, y = +1\} \) and the negative examples \( D^- = \{(x,y) \mid (x,y) \in D, y = -1\} \). At each round \( t \), for step (a), AdaFlock resamples a small number of positive and negative examples, \( S_F^+ \subseteq D^+ \) and \( S_F^- \subseteq D^- \), respectively. The Filter function is called to collect examples until the number of examples in both \( S_F^+ \) and \( S_F^- \) exceed \( \frac{m}{2} \), where \( m \) is the parameter given to AdaFlock. The Filter function performs random sampling without replacement and obtains a sample, \( (x,y) \), and its assigned weight, \( w \). With the probability of \( w \), the Filter function returns the sample which is then added to \( S_F^+ \) or \( S_F^- \). Otherwise the sample is rejected and the Filter function draws another sample.

Next, AdaFlock requests crowdsourcing workers to process feature-definition tasks. In these tasks, the examples in \( S_F^+ \) and \( S_F^- \) are shown to the workers, who are then asked to define the features of round \( F_t = f_1^t, \ldots, f_k^t \), where \( k \) is the number of features of each round. AdaFlock then generates feature-labeling tasks, in which the workers are asked to provide binary labels \( x_{it} = x_{i1}^t, \ldots, x_{ik}^t \) for each object \( x_i \in X \) and features \( F_t \). By using \( \{(x_{it}, y_{it})\}_{i=1}^{N} \), AdaFlock trains a weak learner \( h_t \).

In step (b), similar to the HumanBoost algorithm (Pareek and Ravikumar 2013), AdaFlock computes \( \gamma_t \), which is the weighted accuracy of the weak learner \( h_t \):

\[
\gamma_t = \frac{\sum_{i \in D} 1[y_i = h_t(x_i)] w_i^t}{\sum_{i \in D} w_i^t} - 0.5,
\]

where \( 1[\cdot] \) returns 1 if the condition is true and 0 otherwise. The weighted accuracy \( \gamma_t \) is then used for determining \( \alpha_t \), which is the ensemble weight of \( h_t \):

\[
\alpha_t = \frac{0.5 \log (0.5 + \gamma_t)}{0.5 - \gamma_t}.
\]

Furthermore, step (c) is also performed similar to that in the HumanBoost algorithm. Here, AdaFlock updates the ensemble classifier \( H_t \) by using \( \alpha_t \) and \( h_t \):

\[
H_t(x) = H_{t-1}(x) + \alpha_t h_t(x).
\]

A new weight \( w_{t+1}^i \) is then assigned to each example \( (x_i, y_i) \in D \) by using the outputs of \( H_t \):

\[
w_{t+1}^i = \frac{1}{1 + \exp(y_i H_t(x_i))}.
\]

The assigned weight \( w_{t+1}^i \) is less than 1/2 if \( H_t \) correctly classifies the example and greater than 1/2 otherwise. When \( y_i H_t(x_i) \) decreases, \( w_{t+1}^i \) becomes closer to one so that this example is more likely to be selected for feature-definition tasks in the next iteration.

AdaFlock repeats these steps \( T \) times and outputs the final classifier \( H_T(x) = \text{sign} \left( \sum_t \alpha_t h_t(x) \right) \).

AdaFlock has one termination condition; if the Filter function rejects an example several times, AdaFlock terminates the process and returns the current classifier \( H_t \). This condition is derived from a theoretical guarantee of classification errors. By following HumanBoost, AdaFlock ensures that the error rate becomes sufficiently small after rejecting several samples. Let us define the error rate of \( H_t \) over target distribution \( D \) as \( err_t = \text{Pr}_{(x,y) \sim D} [H_t(x) \neq y] \) and denote \( \epsilon \) a desired error rate. It is guaranteed that \( err_t < \epsilon \) with probability at least \( 1 - \delta_t \) when the Filter function rejects \( n > \frac{2}{\epsilon} \ln \left( \frac{1}{\delta_t} \right) \) samples.

In addition, based on a proposition regarding the number of iterations and the error rate for AdaBoost (Freund and Schapire 1997), we get the number of iterations \( T \) that is sufficient to achieve \( err_t < \epsilon \) as \( T = \frac{2}{\epsilon^2} \ln \frac{1}{\delta_t} \) where the several times to output the final classifier.

2.3 Algorithm

Algorithm 1 shows the detailed procedure of AdaFlock, which is illustrated in Figure 2. We first divide the training dataset into the positive examples \( D^+ = \{(x,y) \mid (x,y) \in D, y = +1\} \) and the negative examples \( D^- = \{(x,y) \mid (x,y) \in D, y = -1\} \). At each round \( t \), for step (a), AdaFlock resamples a small number of positive and negative examples, \( S_F^+ \subseteq D^+ \) and \( S_F^- \subseteq D^- \), respectively. The Filter function is called to collect examples until the number of examples in both \( S_F^+ \) and \( S_F^- \) exceed \( \frac{m}{2} \), where \( m \) is the parameter given to AdaFlock. The Filter function performs random sampling without replacement and obtains a sample, \( (x,y) \), and its assigned weight, \( w \). With the probability of \( w \), the Filter function returns the sample which is then added to \( S_F^+ \) or \( S_F^- \). Otherwise the sample is rejected and the Filter function draws another sample.
error rate of each weak learner $h_t$ is ensured to be less than $1/2$ and $\gamma_* = \min\{\gamma_t\}$.

When we apply the final classifier for a new object $x^*$, we must query feature-labeling tasks for the example by using $\{F_t\}_t$ to create $\{x_t^*\}_t$. The output $h_t(x^*)$ of each weak learner is used to compute the output of the final classifier $H_T(x^*) = \text{sign} \left( \sum_t \alpha_t h_t(x^*) \right)$.

### 3 Experiments

#### 3.1 Experimental Setup

**Datasets** We prepared two datasets for conducting our experiments: *Paintings* and *Smiles*. The paintings dataset contains paintings by two impressionist painters: Claude Monet and Alfred Sisley. We obtained 200 paintings by Monet\(^1\) and 200 paintings by Sisley\(^2\). Each picture was cropped from each side by 30 pixels to remove signatures. The smiles dataset contains videos of spontaneous or posed smiles of enjoyment. We randomly selected 200 spontaneous smile videos and 200 posed smile videos from the UvA-NEMO Smile Database (Dibeklio˘glu, Salah, and Gevers 2012).\(^3\)

Both datasets have 400 examples each. We used 200 examples for training, from which 100 were positive and the others were negative; the remaining 200 examples were used for testing. We prepared two different training-testing splits for each dataset and the methods were performed on each split (i.e., trial).

**Procedure** We fix the number of iterations to a rather large value (i.e., $T = 10$) to investigate the change of the accuracies according to the number of iterations. We set the number of selected examples at each iteration to $m = 20$ and the number of feature definitions at each iteration to $k = 10$ because we consider these settings are convenient for workers. We assumed that $\delta_t$ and $\epsilon$ are sufficiently small values. We employed a decision tree as a weak learner.

We used Lancers,\(^4\) a commercial crowdsourcing platform in Japan, for processing our experiments. In each AdaFlock iteration, feature-definition and feature-labeling tasks were posted to the crowdsourcing platform. We hired $k$ workers for the feature-definition tasks of each iteration. Each worker was shown three positive and three negative examples randomly selected from $S^+_F$ and $S^+_F$, respectively, and was asked to provide one feature definition; thus, we collected $k$ definitions in each round. We obtained a total of $kT$ feature definitions. In the feature-labeling tasks, we asked three crowdsourcing workers to label each object–feature pair, and used the majority voting rule to decide the label. We paid 0.30 USD for defining one feature and 0.05 USD for providing one label for an object–feature pair. For each trial, we spent 630 USD for completing both feature-definition and feature-labeling tasks of $T = 10$ iterations in total.

**Baselines** We prepared three baseline methods to investigate the efficiency of AdaFlock:

- **Crowd prediction**: This method directly asked crowdsourcing workers to classify the samples. Specifically, three crowdsourcing workers were assigned to each of the samples, and their majority voting results were used as the final label.

- **Off-the-shelf features**: We prepared computed features that were not specially designed for a given classification task and trained an AdaBoost classifier by using the features. For the painting dataset, we applied the pre-trained Inception-v3 model (Szegedy et al. 2015) for the images to obtain feature vectors with 1,008 dimensions. We did not prepare off-the-shelf-features for the Smiles dataset because it is not straightforward to extract reasonable features for videos.

- **StandardFlock**: We collected features by using the Flock algorithm (Cheng and Bernstein 2015). We first posted feature-definition tasks to generate $kT$ features. In the feature-definition tasks, each worker was shown three positive and three negative examples randomly selected from the training datasets and was asked to provide a feature definition. We then requested feature-labeling tasks for all the examples and the feature definitions. Unlike the original Flock, we did not apply clustering to organize the obtained feature definitions. Therefore, AdaFlock and this implementation of Flock (called StandardFlock) differ only in adaptive selection of examples for feature-definition tasks. An AdaBoost classifier was trained by using the obtained feature labels.

For each trial of each dataset, StandardFlock and AdaFlock require 630 USD for crowdsourcing. We spent 12 USD for crowd prediction tasks, and Off-the-shelf-features does not require crowdsourcing costs.

#### 3.2 Results

Figures 5a and 5b show the training and testing accuracies at each iteration of AdaFlock, respectively. It is observed that the training accuracies successfully improve with increasing iterations in both the datasets. The testing accuracies increase with increasing iterations as well. It is remarkable that the testing accuracy of the first trial of the paintings dataset at the fifth iteration achieves a 10% improvement over that at the previous iteration. At the fifth iteration, AdaFlock discovered a feature of “is the ground illustrated in this painting?” which was strongly correlated with the true labels and contributed to improve the testing accuracy of AdaFlock. A similar result is observed in the smiles dataset; a feature of “Is the duration of the smile short?” was discovered at an iteration by AdaFlock, which contributed to an 8% improvement in the testing accuracies.

Table 2 shows the testing accuracy of AdaFlock at the end of all iterations in each trial, and those of the baseline methods. AdaFlock outperforms all the baseline methods in most cases. On the paintings dataset, as the performance of the crowd prediction indicates, although crowdsourcing workers did not demonstrate high performances for classifying the pictures, they generated informative features through

\(^1\)http://www.claudemonetgallery.org/
\(^2\)http://www.alfredsisley.org/
\(^3\)http://www.uva-nemo.org
\(^4\)http://www.lancers.jp/
Table 2: Comparison of the testing accuracies for each trial of each dataset. AdaFlock outperforms the baseline methods in most cases.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Trial</th>
<th>Crowd prediction</th>
<th>Off-the-shelf features</th>
<th>StandardFlock</th>
<th>AdaFlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paintings</td>
<td>1</td>
<td>0.560</td>
<td>0.755</td>
<td>0.755</td>
<td>0.820</td>
</tr>
<tr>
<td>Painting 2</td>
<td>2</td>
<td>0.580</td>
<td>0.735</td>
<td>0.745</td>
<td>0.705</td>
</tr>
<tr>
<td>Smiles 1</td>
<td>1</td>
<td>0.720</td>
<td>-</td>
<td>0.715</td>
<td>0.780</td>
</tr>
<tr>
<td>Smiles 2</td>
<td>2</td>
<td>0.715</td>
<td>-</td>
<td>0.720</td>
<td>0.740</td>
</tr>
</tbody>
</table>

Figure 5: Training and testing accuracies at each AdaFlock iteration for each trial of each dataset. The number after each dataset name indicates the trial number. Both the training and the testing accuracies increase with increasing iterations.

AdaFlock. In contrast, on the smiles dataset, whereas the crowd prediction reaches a high level of accuracy, the accuracies of the classifier with StandardFlock are on a similar level. Even in such a case, AdaFlock performs better than both baseline methods.

Additionally, AdaFlock achieves the same level of accuracy as that of StandardFlock in less than or equal to five iterations in three out of the four cases. This indicates that AdaFlock successfully generated more informative features than the baselines at reasonable crowdsourcing costs.

AdaFlock shows a lower accuracy than StandardFlock only in the second trial of the painting dataset. As Figure 5a shows, in this trial, the training accuracy reached to high level in the first few iterations; this indicates that AdaFlock discovered appropriate features for the training samples at these iterations but they slightly overfitted. In such a case, showing more examples in the feature-definition tasks would be efficient to let the crowdsourcing workers generate more general features.

We next investigate the features discovered by AdaFlock and StandardFlock. Table 3 lists the representative features discovered by AdaFlock. It is observed that AdaFlock successfully discovered several informative features that were not discovered by StandardFlock. It is remarkable that AdaFlock generated features related to the ground (e.g., “is the ground illustrated in this painting?”) for the paintings dataset, which were strongly correlated with the true labels but were not discovered by StandardFlock. As this finding indicates, AdaFlock successfully discovers informative features through adaptive selection of difficult examples.

Additionally, we examine the number of redundant features generated by AdaFlock. Considering a feature is redundant if the feature has the same meaning with one of the previously discovered features, we find that 23.5% and 44% of the AdaFlock features are redundant in the painting and smiles dataset, respectively. We can reduce costs for crowdsourced feature generation by avoiding such redundant features and incorporating a procedure for addressing this issue into AdaFlock would be our future work; for example, as AdaFlock is an iterative approach, we can show the features obtained in the previous iterations to workers and ask them to provide features that are not yet collected.

4 Related Work

There have been a few studies addressing crowdsourcing feature discovery. Branson et al. proposed a method for crowdsourced feature labeling (Branson et al. 2010). Although ordinary crowdsourcing workers are not able to recognize detailed categories, such as “Myrtle Warbler” or “Thruston Jackaroo,” which are species of birds, the workers may be able to identify some attributes of examples. Based on this assumption, the authors proposed a framework in which each worker was asked to answer predefined questions, such as “is the belly white?” The answers for each ex-
Table 3: List of generated features for the paintings dataset and the smiles dataset. Features assigned with a high positive or negative correlation coefficient with the true labels of the test samples are shown. Correlation coefficient values are given in brackets. The features not discovered by StandardFlock are bold-faced.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paintings</td>
<td>Is the vast sky illustrated in this painting?; Are clouds illustrated in this painting?; Is the ground illustrated in this painting?; Are people in this painting drawn from a distance?; Are any flowers illustrated in this painting?; Is this painting dreamy rather than realistic?; Is this painted with bold brushwork?; Is this painting abstract?</td>
</tr>
<tr>
<td>Smiles</td>
<td>Does the person keep smiling for more than three seconds?; Does the person shake slightly?; Is the person’s shoulder shaking while they are laughing?; Does the person shake their faces while the corners of their mouths turning up?; Is the duration of smile short?; Does the person suddenly stop smiling?; Does the person turn serious shortly after they stop smiling?; Does the person lean back in a seat while they are laughing?</td>
</tr>
</tbody>
</table>

In this paper, we proposed AdaFlock, an algorithm for generating informative features through crowdsourcing. AdaFlock iteratively obtains helpful features for improving the classification performance. Inspired by AdaBoost, at each iteration of AdaFlock, crowdsourcing workers are shown examples selected according to the classification errors of previous classifiers. They are then asked to generate features useful for correctly classifying the given examples. We apply the FilterBoost approach to sample the examples based on the classification errors. We conducted experiments on a commercial crowdsourcing platform, and the results indicate that AdaFlock successfully discovers informative and interpretable features.

At each iteration of AdaFlock, the positive and negative examples are independently selected based on the classification error of each example; however, the combination of examples would be an important factor for eliciting informative features from crowdsourcing workers. Incorporating such a mutual influence into the algorithm would be the focus of our future work. Combination of AdaFlock and machine learning methods is an interesting research direction that should be pursued. We will consider to replace human labeling process with machine learning methods which use raw image features. If the labeling for a feature is difficult for machines, we can perform additional run of AdaFlock for the sub-classification task. This is another promising future work.

5 Conclusion

In this paper, we proposed AdaFlock, an algorithm for generating informative features through crowdsourcing. AdaFlock iteratively obtains helpful features for improving the classification performance. Inspired by AdaBoost, at each iteration of AdaFlock, crowdsourcing workers are shown examples selected according to the classification errors of previous classifiers. They are then asked to generate features useful for correctly classifying the given examples. We apply the FilterBoost approach to sample the examples based on the classification errors. We conducted experiments on a commercial crowdsourcing platform, and the results indicate that AdaFlock successfully discovers informative and interpretable features.

At each iteration of AdaFlock, the positive and negative examples are independently selected based on the classification error of each example; however, the combination of examples would be an important factor for eliciting informative features from crowdsourcing workers. Incorporating such a mutual influence into the algorithm would be the focus of our future work. Combination of AdaFlock and machine learning methods is an interesting research direction that should be pursued. We will consider to replace human labeling process with machine learning methods which use raw image features. If the labeling for a feature is difficult for machines, we can perform additional run of AdaFlock for the sub-classification task. This is another promising future work.

References


