分類マイクロタスクにおけるタスク順序制御手法

根本千代之介 †　　田島　敬史††　　森嶋　厚行†††

† 筑波大学大学院図書館情報メディア研究科　〒 305–8577 茨城県つくば市天王台 1 丁目 1 番 1 号
†† 京都大学大学院情報学研究科　〒 606–8501 京都市左京区吉田本町
††† 筑波大学 知的コミュニティ研究センター　〒 305–8550 茨城県つくば市春日 1 丁目 2 番地 1 号
E-mail: †s1521637@u.tsukuba.ac.jp, ††tajima@i.kyoto-u.ac.jp, †††mori@slis.tsukuba.ac.jp

あらまし　クラウドソーシングにおける典型的なタスクの一つにデータの分類がある．一般に，クラウドソーシングでは多くのワーカの存在を仮定する事が多いものの，タスク数が非常に多い場合など，実際には，必ずしも短時間で全てのタスクが処理されるとは限らない．本論文では，災害時に置ける災害箇所の発見など，あるクラスのデータを発見するケースに関して，できるだけ早く多くのデータを発見するためのタスク順序の制御手法について議論する．具体的には統計情報を利用した方法と機械学習を用いた方法を比較した．その結果, フィルタ数が多いとき, 機械学習を用いた手法よりも統計情報を利用した手法の方が再現率の立ち上がりにおいて優れている場合があることを発見した．

キーワード　クラウドソーシング, 分類, タスク割り当て制御

1. はじめに

計算機だけでは解決が困難な問題に対するアプローチとして，クラウドソーシングが多くの分野で用いられている．

クラウドソーシングの一形態として, 短時間で処理可能なタスク (マイクロタスク) を用いるマイクロタスク型クラウドソーシングがある．マイクロタスクの例を図 1 に示す．図 1 は航空写真に写っている家屋が倒壊しているか否かを判別するマイクロタスクである．このような，分類を行うためのタスクを本論文では分類タスクと呼ぶ．

本論文では，大量の分類タスクに対して，十分な数のワーカが存在せず，タスクを順次処理していかなければならないような状況において，どのような順序でタスクを処理すべきかという問題について議論する．そのような問題が重要な応用はいくつか存在する．そのような例を次に示す．(例 1) 被災地の救援を目的として，航空写真群に写っている被災家屋を見つけたい場合, 最終的には全ての被災家屋を発見したいとしても, できるだけ早く多くの被災している箇所を発見したい．(例 2) データセットの中から，条件を見たす少数の例を発見できればよく，かつ, データセットの量が膨大である場合には，ワーカに支払う報酬の合計額を少なくするために, できるだけ少ないタスクでそれらの例を発見したい．

タスクの処理順序を決定するために，各タスクが扱うデータの特徴に関するヒューリスティクスを利用するというアプローチが考えられる．これは，具体的には特徴量に関する真偽値を返す判定という形で表現されるため，本稿では**特徴量フィルタ** (以下, フィルタとよぶことがある) と呼ぶ．例えば, 土砂崩れの被災地の航空写真群から多数の倒壊家屋を発見する場合, 茶色がかった箇所から優先してタスクを処理することにより, より少ないタスク数で多数の倒壊家屋を発見できると予想される．この例では，「画像が茶色がかっているれば真」というフィルタを利用し, その条件にマッチしたデータを持つタスクを優先して処理する．

図 1　マイクロタスクの例

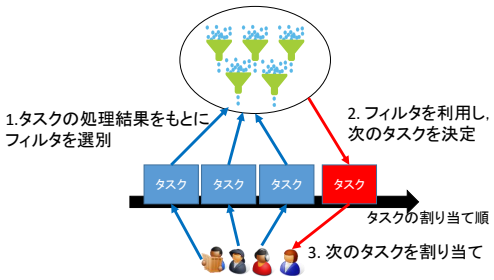

図 2　フィルタを利用した, タスク処理順序決定手法

しかし, どのフィルタが効果的であるかどうかはデータセットの性質に依存する．例えば, 土砂崩れの被災地と津波の被災地では, 同じ図 1 のような被災家屋発見タスクでも有効な特徴量フィルタは異なると予想される．このように, どのフィルタを適用すべきかはデータセットの性質に応じて判断する必要がある．しかし, あらかじめどのフィルタが有効であるかどうかを調査する作業は必ずしも容易ではなく, その作業により, タスクの依頼開始が遅れる可能性がある．

そこで, 本稿では, あらかじめ大量のフィルタを用意し，その中から効果的なものを動的に取捨選択しながら適用しタスクの順序を決定する手法について述べる．図 2 にこの手法の概要を示す．近年, クラウドソーシングを用いて有効そうな特徴量を発見する研究が行われており [7] [4], このような手段で入手した大量のフィルタを利用するというアプローチは必ずしも非現実的ではない．具体的には，統計情報を利用して取捨選択を行う手法と，既存の機械学習アルゴリズムを用いて取捨選択をする手法について述べ, それらを比較し，それぞれの特性を明らかにする．

本論文の貢献は次のとおりである．

(1) クラウドソーシングの中間結果に注目した研究．これまで，クラウドソーシング結果の品質に関しては, 個々のタスクの結果や, 最終的な結果の品質を対象としていた．しかし, 一般にクラウドソーシングは全タスクの終了までに時間がかかるため, 中間時点でのデータ品質も重要と考えられる．本研究は中間時点でのデータ品質に着目して, タスク割り当て順の制御を行うものである．また, これまで情報検索の分野において，検索のランキングの評価などにおいて，上位に望ましい結果が来ることを目的とする研究が数多く行われてきた．本研究は，そのような視点を，クラウドソーシングにおけるタスクの割り当て順制御に導入しようとするものである．

(2) 実データを用いた複数手法の比較．本論文では，実データを用いて，統計情報に基づく手法と機械学習に基づく手法の比較を行った．その結果, 大量のフィルタが存在する場合には，次元の呪いによって機械学習の学習に時間がかかるため，統計情報に基づく手法が，再現率の立ち上げにおいて優れている場合があることを発見した．

なお, 本研究では, ワーカはタスクに対し必ず正しい回答をすると仮定する．実際, これは現実的ではないが, タスク結果の品質向上に関する研究はすでに行われていて, それらで述べられている手法を用いれば, 比較的高い確率で正しい回答が得られる．そのため, 本研究では, タスク結果の品質については議論の対象としない．

本稿の構成は以下のようになる．第 2. 節では関連研究について述べる．第 3. 節ではヒューリスティクスのモデル化に用いるフィルタについて述べる．第 4. 節では, 本論文で扱う問題について形式的に述べる．第 5. 節では統計情報を用いたタスク割り当てアルゴリズムについて説明する．第 6. 節では機械学習を利用したタスク割り当てアルゴリズムについて説明する．第 7. 節では実験とその結果について述べる．第 8. 節はまとめである．

2. 関 連 研 究

本論文と同様の課題設定での研究 [8] と比較して，本論文は統計情報に基づく手法のアルゴリズムが異なり，また，既存の機械学習アルゴリズムとの比較を行っているところに新規性がある．その他の関連研究は次の通りである．

Multiple Classifier System (MCS) 本手法におけるフィルタの選別と関連する研究として，MCS [6] がある．MCS に関する研究は，複数の Classifier を用いて，よりよい分類結果を得るためのものである．MCS に関する研究は複数のアプローチで行われているが，本手法に最も関連するアプローチとして Classifier Selection がある．Classifier Selection は複数の Classifier から，優れている Classifier を選ぶ問題であり, 我々が複数のフィルタから, 優れたフィルタを選ぶのと本質的には同じ構造を持つ．

しかし，本研究における「優れた特徴量フィルタ」と, 一般に Classifier Selection における「優れた Classifier」では選択の基準が異なる．Classifier Selection が選別の尺度として主に正確率 (Accuracy) を用いるのに対し，我々は，分類タスクにおいて早く正例を見つけることを問題にしているため，フィルタの選別に適合率 (Precision) を用いる．そのため，フィルタの選別では, フィルタにマッチするデータのみを収集すれば十分であり, フィルタにマッチしないデータについては考慮する必要が無い．例えば, 赤いデータにマッチするフィルタと黄色いデータにマッチするフィルタの間で適合率の比較を行う場合, 赤いデータと黄色いデータだけ収集すればよい．

また, 与えられたデータセットに対して正しそうな Classifier を予測して選択するような Classifier Selection は Dynamic Classifier Selection と呼ばれている．Dynamic Classifier Selection の研究としては, [5] がある. [5] はトレーニング済みのデータから分類するデータに近いものをいくつか集め，それに対する正確度を用いて，Classifier を選別する手法を提案している．Dynamic Classifier Selection は全体の候補からデータセットに合わせて適したものを選ぶという点で本研究と共通する．しかし，Dynamic Classifier Selection は Classifier の選別と適用の 2 フェーズに分かれているのに対し，本手法はフィルタの選別と適用を 1 フェーズで行うものである．

群衆によるフィルタの提案と作成 群衆にフィルタの提案や作成を行わせた研究として Cheng らの研究 [4] や, Zou らの研究 [7] がある．これらの研究は, 人間に判定の手がかりとなる特徴の発見と特徴量の付与を行わせ, 人間が作成した特徴量と機械的に作成した特徴量を組み合わせた分類器を作成するものである．これらの研究では, 特徴の発見とラベル付けを人間に行わせている．これは本研究におけるフィルタの発見と作成に相当する．

これらの研究ではフィルタの発見と作成を不特定多数のワーカに行わせているのに対し, 本研究ではどちらもリクエスタ (タスクを作成する人) 自身が機械的な手法を用いて行う．したがって, 本研究は, 機械的に導出できるようなフィルタを発見する必要があるという点で, これらの研究と比べ適用範囲が狭い．しかし, そのかわり, 人間に特徴量を付与させることを必要としない．

3. 特徴量フィルタ

本節では, ヒューリスティクスのモデル化に使用する特徴量フィルタを定義する．なお，以降では単に「フィルタ」と呼ぶことがある．

特徴量フィルタ f_i とは， データ d_jが性質 p_i を満たせば 1 を, そうでなければ 0 を返す関数である．例えば，茶色い画像にマッチするフィルタ $f_{brown}(d_j)$ は, d_j が茶色い画像データで

あれば 1 を, そうでなければ 0 を返す. 形式的には次のように定義される.

定義 3.1 (特徴量フィルタ). $D = \{d_1, \ldots, d_n\}$ をデータ集合とし, p_i を判定したい性質とする. このとき, あるデータ $d_j \in D$ が性質 p_i を満たすか否かを判定する特徴量フィルタは, 関数 $f_i \colon D \to \{1, 0\}$ である. ただし,

$$f_i(d_j) = \begin{cases} 1 & \text{if } d_j \text{ satisfies } p_i \\ 0 & \text{if } d_j \text{ dosen't satisfy } p_i \end{cases}$$

空フィルタ f_{emp} は全てのデータにマッチする特徴量フィルタであり, 次のように定義される.

定義 3.2 (空フィルタ). 空フィルタ $f_{emp} : D \to \{1, 0\}$ は次である.

$$(\forall d_i \in \mathrm{D})\ f_{emp}(d_i) = 1$$

次に, 複数の特徴量フィルタを用いて, 特徴量ベクトルを定義する.

定義 3.3 (特徴量ベクトル). 性質の列 $p_1 \ldots p_m$ に関するデータ $d_i \in D$ の特徴ベクトル $\boldsymbol{x_i}$ とは次である.

$$\boldsymbol{x_i} = (f_1(d_i), f_2(d_i), \cdots, f_m(d_i))$$

4. 本論文で扱う問題

本論文で扱う, 分類タスクの割当て順序制御の問題は次の通りである. まず, データ集合 $D = \{d_1, d_2 \ldots, d_n\}$ 中の各データが, クラス c に属するか否か分類するためのタスク集合 $T = \{t_1, t_2, \ldots, t_n\}$ を用意する. ここで, 各 t_i は d_i がクラス c に所属するか否か分類するためのタスクである (図 1).

このとき, 本論文における分類タスクの割当て順序制御とは, タスクの処理順序を表す列 T' を動的に作る事である. ここで, T' は T 中の全てのタスクをそれぞれ一度含む列である. 例えば, $T = \{t_1, t_2, t_3\}$ の時, $T' = [t_2, t_1, t_3]$ などが考えられる.

T' を**動的に作る**とは, T' を先頭から順に作る際に, 時点 k までに割り当てたタスク列 T'_k (最終的な T' の部分列) のタスク結果を見て, 次のタスク t_p を決定すると言う事である (図 2). また, 本論文では, 次のタスクの決定の際に, 与えられた (複数の) 特徴量フィルタを利用する. したがって, 本論文で扱う手法は, 次の手順になる.

（1） 入力として, データ集合 $D = \{d_1, \ldots, d_n\}$, タスク集合 T, および特徴量フィルタの集合 $F_{all} = \{f_1, \ldots, f_m\}$ をとる.

（2） 時点 0 でのタスク列を $T'_0 = []$ とする.

（3） タスクを行ってもらいながら, 時点 $k+1$ のタスク列 T'_{k+1} を順次計算する.

（4） $T'(= T'_n)$ と, T' に含まれる分類タスクを処理した結果の列 $Results$ を出力する.

上記 (3) において T'_{k+1} を求める際の入出力は次の通りである.

入力 ある時点 k までに行われたタスク列 T'_k と, T'_k に含まれる分類タスクを処理した結果 $Results_k$, および特徴量フィルタの集合 $F_{all} = \{f_1, \ldots, f_m\}$.

出力 $T'_{k+1} = T_k + [t_p]$. ただし, t_p は, **特徴量フィルタを用いて**, $T - T'_k$ から選ばれる.

出力される T' の善し悪しは, いかに少ないタスク数で多くの正例 (クラス c に属するデータ) を発見できたかで評価される.

以降では, 統計情報を利用したアルゴリズムと, 既存の機械学習手法を利用したアルゴリズムを説明する. これらは, 上記 (3) において, t_p を選ぶために利用する特徴量フィルタをどう選別するかの部分が異なる.

5. 統計情報を利用したタスク割り当てアルゴリズム

統計情報を利用した手法では, 時点 k までのタスク処理結果に関する特徴量フィルタの統計情報をもとに, t_p を選択するために利用するフィルタの選別を行う.

5.1 概　　要

統計情報を利用した手法のアルゴリズム SFSelect (Statistics based Filter Select) を Algorithm 1 に示す. SFSelect の入力はタスク集合 T とフィルタ集合 F_{all}であり, 出力はタスク処理結果の列 $Results$ である.

このアルゴリズムにおけるフィルタの選別手法は, Backward stepwise selection の一種である. 具体的には, F_{all} 中のフィルタをまず F_{active} というフィルタ集合に所属させ, 他のフィルタと比べて相対的に有効でないと一時的に判断されたフィルタを順次 $F_{inactive}$ に移動させる. また, マッチするデータを全て見つけたフィルタは F_{finish} というフィルタ集合に移動させる (図 3). F_{finish} の詳細は, 5.2 節で説明する.

本アルゴリズムは次のように動作する. まず, F_{all} 中のフィルタを全て F_{active} に所属させ, $F_{inactive}$, F_{finish} は空にする (1 行目, 2 行目) . つまり, 全てのフィルタが有効なものであるとみなす.

そして未処理のタスクのうち, F_{active} に属するフィルタに最も多くマッチするデータを持つタスクを選ぶ (5 行目) . 5 行目の getMaxMatchTasks 関数は, 未処理のタスク集合から, F_{active} に属するフィルタに最も多くマッチするタスクの集合を返す関数である. マッチするフィルタの数が多いデータを持つタスクを優先して割り当てるのは, マッチするフィルタが多いほど, 正例である確率が高いと期待するためである. getMaxMatchTask 関数から返された結果からワーカに割り当てるタスクをランダムに 1 つ選択する (11 行目) . そして選択されたタスクをワーカに割り当て, 回答を得る (12 行目) .

ワーカから回答が得られたら, その結果を $Results$ に加え (13 行目) , タスク処理結果保存テーブル RTable の内容を更新する (14 行目). RTable の例を表 1 に示す. RTable を利用してフィルタの選別を行い, 再びタスクの処理を開始する.

5.2 フィルタの選別

本節ではフィルタの選別方法 (15 行目〜21 行目) について説明する. フィルタの選別は F_{active} に属するフィルタに対して,

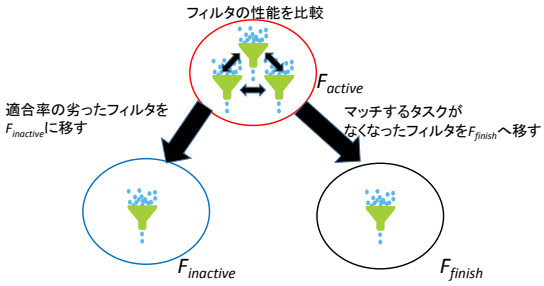

図 3　3 つのフィルタ集合とフィルタ選別

表 1　RTable の例

フィルタ名	正例	負例
f_1	1	10
f_2	8	1
f_3	4	4
f_4	5	2

RTable を用いて行う (15 行目) . そして選別の結果, 取り除くフィルタを F_{active} から $F_{inactive}$ に移す. この動作は図 3 の左側に示されている.

具体的には, F_{active} に属する 2 つのフィルタの組み合わせ全てについて, 適合率に有意差があるか検定を行う. 17 行目の existSignificantDiff 関数は, 与えられた 2 つのフィルタについて検定を行い, 2 つのフィルタの適合率に有意差が有るか否かを返す関数である. この関数は RTable の値をもとに検定を行う. RTable には各フィルタにマッチしたタスクに関して, 正例の個数と負例の個数が保存されており, これを用いてフィルタの適合率を計算する. 検定の結果, 2 つのフィルタの適合率に有意差が存在するとされた場合, 適合率が劣っている方のフィルタを F_{active}から $F_{inactive}$ に移す (18 行目, 19 行目). つまり, 適合率が有意に劣ったフィルタを, タスク処理順の変更に用いるフィルタの集合 (F_{active}) から取り除く. 18 行目の getInferiorF 関数は 2 つのフィルタを引数としてとり, 適合率が低い方のフィルタを返す関数である.

仮説検定の手法としては, フィッシャーの正確確率検定とカイ二乗検定を用い, Precision(f_1) = Precision(f_2) という帰無仮説を検定する. フィッシャーの正確確率検定は標本数が少ない場合にも正確な検定が行えるが, 標本数が大きくなると計算量が膨大になることが知られている. そのため本手法では, 標本数が少ない場合にはフィッシャーの正確確率検定を用い, 多い場合にはカイ二乗検定を用いる.

5.3　割り当てるタスクがない場合の処理

SFSelect では, ワーカに割り当てるタスクが存在しない場合が生じる. その場合の処理 (6 行目から 9 行目) について説明する.

まず, 割り当てるタスクが存在しないとは, F_{active} 中のフィルタにマッチするデータをもつ, 未処理のタスクが存在しないということである. すなわち, (現時点において) 有効なフィルタにマッチするデータを持つタスクを全て処理し終えている状況を指す.

Algorithm 1 SFSelect

Input : T, F_{all}
Output : *Results*

```
1:  F_active ⇐ F_all
2:  F_inactive ⇐ ∅, F_finish ⇐ ∅
3:  Results ⇐ []
4:  while |Results| != |T| do
5:      candidateTasks ⇐ getMaxMatchTasks(T, F_active )
6:      if candidateTasks.empty? then
7:          F_finish ⇐ F_finish ∪ F_active
8:          F_active ⇐ F_inactive
9:          next
10:     end if
11:     task ⇐ getRandom(candidateTasks)
12:     result ⇐ ask(task) //タスクをワーカに割り当て, 回答を入手
13:     Results.add(result)
14:     updateTable(RTable, Results)
15:     for all f1, f2 s.t. f1 ∈ F_active ∧ f2 ∈ F_active ∧ f1 ≠ f2 do
16:         /*フィルタの性能をペアワイズで比較*/
17:         if existSignificantDiff(RTable, f1, f2) then
18:             inferiorF ⇐ getInferiorF(f1, f2)
19:             F_active.delete(inferiorF), F_inactive.add(inferiorF)
20:         end if
21:     end for
22: end while
```

この場合, F_{active}中のフィルタの適用を終了し, $F_{inactive}$ に属するフィルタを対象に選別を開始する. つまり, 残っているフィルタ集合を対象に, フィルタの選別を再び始める. 具体的には, まず, 図 3 の右側のように, F_{active} 中のフィルタを全て F_{finish} へと移す (7 行目) . F_{finish} は, 未処理のタスクのいずれにもマッチしないフィルタの集合である. そして, $F_{inactive}$ 中のフィルタを全て F_{active} に移し (8 行目) , 再びタスクの処理とフィルタの選別を開始する.

5.4　SFSelect の拡張

SFSelect を拡張することにより, 「他の結果に依存するフィルタ」も扱えるようになる. 「他の結果に依存するフィルタ」とは, 例えば, 図 1 のような被災家屋判定タスクの場合, 「壊れている家屋が周囲 100m 以内に存在する」といったものである. この例の場合, 他のタスクが処理され, 壊れている家屋が発見された場合, その家屋の周囲 100m 以内にある画像がこのフィルタにマッチするようになる. すなわち, 他のタスクの処理結果によって, あるデータに対するフィルタの出力が変化する.

「他の結果に依存するフィルタ」を扱えるよう SFSelect を拡張するには, RTable を更新するタイミングを増やすことと, 一度 F_{finish}に所属させたフィルタを F_{active} へ戻すことが必要となるが, ここではその詳細は省略する. 詳しくは権守らの論文 [8] を参照されたい.

6. 機械学習を用いたタスク割り当てアルゴリズム

本節では, 機械学習を用いたタスク割り当てアルゴリズム ML について説明する. このアルゴリズムでは各フィルタを特徴量とみなし, タスクの処理結果をもとに各フィルタに重みづけを行う. そしてフィルタの重みをもとに各タスクのスコアを計算し, スコアが最も高いタスクをワーカに割り当てる.

機械学習を用いたタスク割り当てアルゴリズム ML を Algorithm2 に示す. このアルゴリズムの入力は, 統計情報を利用した手法のアルゴリズムと同じく, タスク集合 T とフィルタ集合 F_{all} である. 出力はタスク処理結果の列 $Results$ である.

ML では, ワーカからの回答を訓練データと見なし, 訓練データをもとに学習を行う. そして学習結果をもとに各フィルタに対し重みづけを行う (8 行目). 訓練データは第 3. 節で述べた特徴ベクトルと同じ形式をとり, ワーカからの回答を正解ラベルとして持つ. そして得られた重みを用いて未処理のタスク全てのスコアを計算し, スコアが最も高いタスクをワーカに割り当てる (4 行目, 5 行目, 6 行目). スコアが高いほど, 正例である確率が高いとする. 4 行目の getMaxScoreTask 関数は未処理のタスク全てについてスコアを計算し, スコアが最も高いタスクの集合を返す関数である. なお, スコアが最も高いタスクが複数存在した場合, それらのタスクからワーカに割り当てるタスクをランダムに 1 つ選択する (5 行目) .

ワーカから新たな回答が得られると (6 行目), そのデータを訓練データに加え (8 行目) 再学習を行い, 各フィルタの重みを更新する (9 行目). 以上のようなステップを全てのタスクを処理するまで繰り返す.

疑似コード中の Classifier に対応する分類器としてはロジスティック回帰分析を用いた. Support Vector Machine 等の別の分類器を用いることも可能ではあったが, 今回は簡単のため, ロジスティック回帰分析を用いた. 具体的には統計解析ソフト R の glm 関数を用いた.

glm 関数を用いると, 各タスクのスコアは次のように計算できる. すなわち, まず, glm 関数に引数として訓練データの集合をわたすと, 各フィルタの重みが返される. このとき, あるタスク t において使用するデータを d とするとき, t のスコア $score(t)$ は以下のように計算される.

$$score(t) = w_1 f_1(d) + w_2 f_2(d) + \cdots + w_m f_m(d), \quad w_i \in R, f_i(d) \in \{0, 1\}$$

ここで, w_iはフィルタ f_iの重みであり,実数である. 例えば, フィルタ数が 3 でそれらの重みが $(w_1, w_2, w_3) = (2, 1, 1)$ であり, あるタスク t に対応するデータ d ついて $(f_1(d), f_2(d), f_3(d)) = (1, 0, 1)$ であるとき,t のスコアは

$$score(t) = w_1 f_1(d) + w_2 f_2(d) + w_3 d_3(d) = 2*1+1*0+1*1 = 3$$

となる

Algorithm 2 ML

Input : T, F_{all}

Output : $Results$

```
1: TrainingData ⇐ ∅
2: Results ⇐ ∅
3: while |Results| != |T| do
4:     candidateTasks ⇐ Classifier.getMaxScoreTask(tasks)
5:     task ⇐ getRandom(candidateTasks)
6:     result ⇐ ask(task) //タスクをワーカに割り当て, 回答を入手
7:     Results.add(result)
8:     TrainingData.add(result)
9:     Classifier.learn(results, F_all) //現在までのタスク処理結果を
       利用し再学習
10: end while
```

7. シミュレーション

本節では, SFSelect と ML の差異を調査するために行ったシミュレーションについて述べる. シミュレーションでは 2 種類のデータを利用した. 1 つは E メールのスパム判定データであり, もう 1 つは台風被害の実データである. E メールのデータについては, フィルタ数の影響を調査するため, フィルタ数が 100 の場合と 200 の場合それぞれについてシミュレーションを行った.

なお, SFSelect と ML はいずれも結果がランダムな要素の影響を受けるため, それぞれ 3 回実験を行い, 3 回のデータの平均を求めた.

7.1 問題 1. E メールのスパム判定

7.1.1 シミュレーション設定

CS MINING GROUP [1] にある, 多数のメールとそれらがスパムであるか否かを判定したデータを用いた. データの件数は 4,327 件であり, そのうちの約 31.8% (1,378 件) がスパムと判定されている.

なお, フィルタを選別する際, 有意水準 0.05 の両側検定を行った.

7.1.2 用いたフィルタ

空フィルタも含め, 100 個または 200 個のフィルタを用いた. 用いたフィルタの一部を表 2 に示す. 表 2 に示した以外のフィルタもこの 5 つのフィルタと同様のものである. ただし, 単語の部分をそれぞれ変更している.

フィルタの作成に用いる単語としては, GSL (General Service List) [2] の単語を頻度順に並べた際の上位の語を用いた. ただし, CoreNLP のストップワードリスト [3] を用いて事前にストップワードを除去した. フィルタ数が 100 の場合には上位 99 語を, フィルタ数が 200 の場合には上位 199 語を用いてフィルタを作成した.

7.1.3 シミュレーション結果

図 4, 図 5, 図 6, 図 7 にシミュレーション結果を示す. これらの図は横軸が処理されたタスク数, 縦軸が正例に対する再現率であり, タスク数の増加に伴う再現率の推移を示している. 図 4, 図 5 はフィルタ数が 100 の場合の結果であり, 図 6, 図 7 は

表 2　問題 (1) のシミュレーションに用いるフィルタ (一部)

	内容	適合率	再現率
f_{emp}	（空フィルタ）	31.8%	100%
f_1	本文に「church」が含まれている	85.1%	2.9%
f_2	本文に「public」が含まれている	84.5%	42.8%
f_3	本文に「name」が含まれている	83.9%	48.1%
f_4	本文に「city」が含まれている	76.6%	5.5%

フィルタ数が 200 の場合の結果である．また, 図 4, 図 6 は全てのタスクが終了するまでの結果を示したものであり, 図 5, 図 7 は 500 タスク目までの途中結果を示したものである．本研究では, 少ないタスク数で多くの正例を収集したいという状況を想定しているため, 処理されたタスク数が少ない時点での, 正例に対する再現率が重要となる．

図 4, 図 6 より, SFSelect と ML はいずれも, ランダムな順番でタスクを処理する場合よりも少ないタスク数で比較的多くの正例を発見できていることが分かる．

また, 図 5, 図 7 より, SFSelect と ML とでは, 約 400 タスク目までではSFSelect の再現率が ML のそれよりも高くなっていることが分かる．

さらに, 図 5, 図 7 を比較すると, フィルタ数を増やした場合, タスク数が少ない時点において, 両手法の差が大きくなっていることが分かる．特に, 200 タスク目を処理した時点での両手法の再現率の差は, フィルタ数が 200 の場合の方が 100 である場合と比べ大きくなっている．これは, ML が次元の呪いの影響を受けたためであると予想される．この実験では 200 個のフィルタ, すなわち 200 個の特徴量を用いた．そのため ML では次元の呪いの影響を受け, フィルタの重みを学習するのに多くの訓練データを要し, 再現率の上昇が鈍くなったと予想する．

一方, 図 4, 図 6 から, 約 400 タスク目以降においては ML の方が常に再現率が高くなっていることが分かる．これは (1) 分類器の学習に必要な訓練データがある程度集まったことと, (2) 両手法の間でフィルタに対する重みのつけかたが異なっていたことが原因であると予想する．以下では (2) について詳しく説明する．SFSelect では, あるフィルタに対し, そのフィルタを使用するか否か, すなわち 0 か 1 かという離散的な重みづけを行う．一方, ML では, あるフィルタに対し, 連続的な実数値の重みづけを行う．そのため, ML は SFSelect と比べ, より詳細にフィルタ間の優劣を決定でき, 正例である確率がより高いデータを持つタスクを選択できる．このような理由から, タスク数が増加するにつれ, ML の方が SFSelect よりも再現率が高くなったと予想する．

7.2　問題 2. 建物の倒壊判定

7.2.1　シミュレーション設定

2013 年に発生した台風 26 号の伊豆大島における台風被害の実データを用いた．この実データは 11,350 件の画像からなっており, その内の約 1% (118 件) に倒壊家屋が含まれている．なお, 家屋が倒壊しているか否かの判定は事前に人手で行った．このデータから図 1 のような分類タスクが作成される．

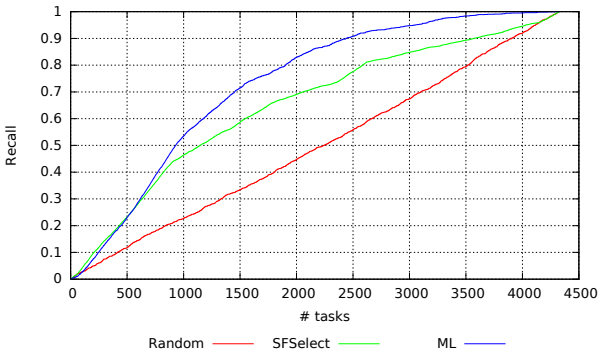

図 4　再現率の推移 (E メール, フィルタ数 100, 全体)

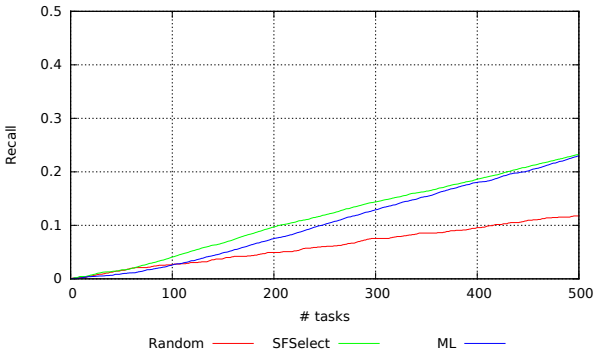

図 5　再現率の推移 (E メール, フィルタ数 100, 500 タスク目まで)

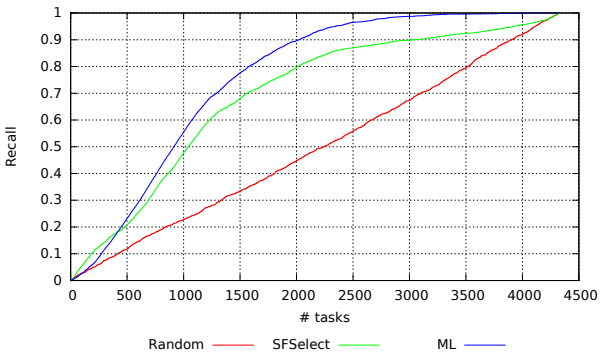

図 6　再現率の推移 (E メール, フィルタ数 200, 全体)

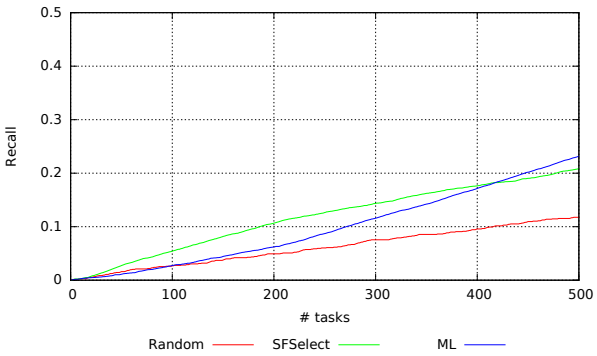

図 7　再現率の推移 (E メール, フィルタ数 200, 500 タスク目まで)

7.2.2　用いたフィルタ

表 3 に示す 4 つのフィルタを使用した．各フィルタの作成方法については本稿における議論と関係しないため省略する．な

表 3　問題 2. のシミュレーションに用いるフィルタ

	内容	適合率	再現率
f_{emp}	（空フィルタ）	1.0%	100%
f_1	建物周辺の画素の色がまばらである	0.9%	44%
f_2	建物周辺の画素の多くが茶色である	4.8%	53.4%
f_3	壊れている建物の 100m 以内にある	28%	85.6%

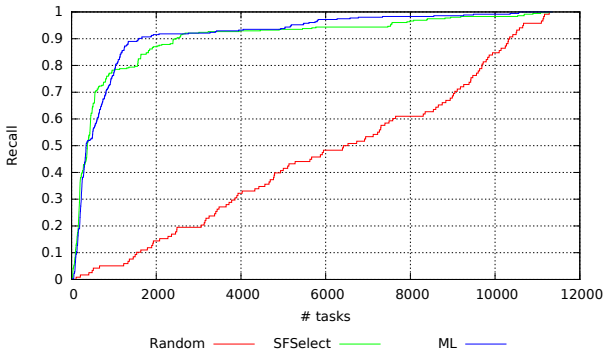

図 8　再現率の推移 (建物の倒壊判定, 全体)

お, 表 3 にあるフィルタ f_3 は他のタスクの結果に依存するフィルタである.

7.2.3　シミュレーション結果

図 8, 図 9 にシミュレーション結果を示す. 図 8 は全 11,350 タスクにわたる結果であり, 図 9 は 1,000 タスク目までの結果である.

図 8 より, SFSelect と ML のいずれも, タスク数が少ない時点でも, ランダムな順でタスクを処理する場合と比べ再現率が高くなっていることが分かる. 特に, 図 9 から, どちらの手法も 1,000 タスクの処理を終えた時点で再現率が 0.8 近くに達していることがわかる. 一方, ランダムな順でタスクを処理する場合では, 1,000 タスクの処理を終えた時点での再現率が 0.1 未満となっている.

これは両手法とも, 他のフィルタと比べ適合率が比較的高い f_2, f_3 を重視したためである. 実際, SFSelect では, f_3 にマッチするデータが存在する時点では, 適合率の最も高い f_3 だけを F_{active} に所属させることが多く, f_3 にマッチするデータが存在しない時点では, 2 番目に適合率の高い f_2 だけを F_{active} に所属させることが多かった. また, 図 10 から分かるように, ML では約 50 タスク目以降において f_2 と f_3 に比較的大きな重みを付与している. 図 10 の横軸は処理されたタスク数, 縦軸は各フィルタに付与されて重みであり, タスク数の変化に伴う各フィルタの重みの変化を示している. この図から, f_2, f_3 に対し, 比較的大きな重みが付与され, f_2, f_3 にマッチするデータを持つタスクが優先的に処理されたことがわかる.

一方, SFSelect と ML では, タスク数に関わらず, 顕著な差は見受けられなかった. 顕著な差がみられなかった要因としては, フィルタ数が少なかったことが予想されるが, これを確認するためのデータは現時点で得られていない. そのため, このような結果となった要因について, さらに実験と分析を行う必要がある.

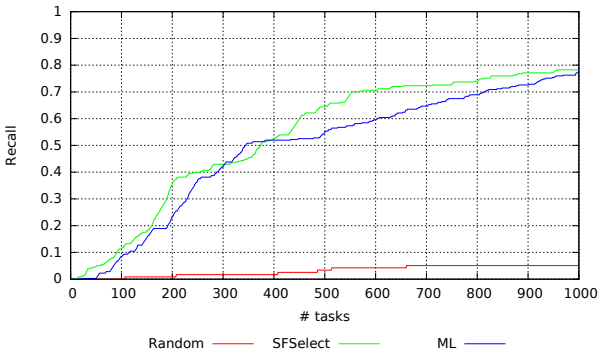

図 9　再現率の推移 (建物の倒壊判定, 1,000 タスク目まで)

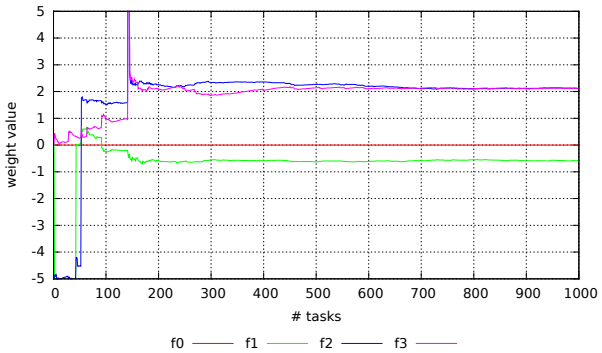

図 10　各フィルタにつけられた重みの推移 (1,000 タスク目まで)

8.　おわりに

本研究では, 大量のデータ中から少ないタスク数で多くの正例を発見することを目的とし, 大量のフィルタ中から選別したフィルタを用いてタスクの割り当て順を変更する問題を扱った. そして, 実データを用いたシミュレーションにより, 機械学習を利用した手法と統計情報を用いた手法の比較を行った. シミュレーションの結果, フィルタ数が多いときには, 機械学習を用いた手法よりも統計情報を利用した手法の方が再現率の立ち上がりにおいて優れている場合があることを発見した.

今後の課題としては，第一に，今回用いたデータ以外での実験がある．次元の呪いにより機械学習の立ち上げが遅れるという現象が，どのような条件でより顕著に出現するのかを検討する．また，今回はフィルタの作りやすさから E メールのスパム判定のデータセットを利用したが，本研究で想定する「立ち上げが重要」という応用で利用されるデータでの実験をさらに追加する．第二に，計算機で判断可能な大量のフィルタを入手する方法が存在すれば，本アプローチが有効な場面が増えると考えられるため，そのような手法の検討が重要である．

謝　　辞

本研究の一部は科研費 (#25240012) による.

文　　献

[1] http://csmining.org/index.php/spam-email-datasets-.html.
[2] http://jbauman.com/aboutgsl.html.
[3] https://github.com/stanfordnlp/CoreNLP/blob/master/data/edu/stanford/nlp/patterns/surface/stopwords.txt.

[4] Justin Cheng and Michael S. Bernstein. Flock: Hybrid crowd-machine learning classifiers. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW 2015, Vancouver, BC, Canada, March 14 - 18, 2015*, pp. 600–611, 2015.

[5] Kevin S. Woods, Kevin W. Bowyer, and W. Philip Kegelmeyer. Combination of multiple classifiers using local accuracy estimates. In *1996 Conference on Computer Vision and Pattern Recognition (CVPR '96), June 18-20, 1996 San Francisco, CA, USA*, pp. 391–396, 1996.

[6] Michal Wozniak, Manuel Graña, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, Vol. 16, pp. 3–17, 2014.

[7] James Y. Zou, Kamalika Chaudhuri, and Adam Tauman Kalai. Crowdsourcing feature discovery via adaptively chosen comparisons. In *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2015, November 8-11, 2015, San Diego, California.*, p. 198, 2015.

[8] 権守健嗣, 森嶋厚行, 歳森敦, 北川博之. クラウドソーシングヒューリスティクスの一般化選択フィルタによるモデル化と動的選択. DEIM2015 第 7 回データ工学と情報マネジメントに関するフォーラム, 2015.