Simpson, E., & Roberts, S. (2015). Bayesian Methods for Intelligent Task Assignment in Crowdsourcing Systems. In T. Guy, M. Kárný, & D. Wolpert (Eds.), *Scalable Decision Making: Uncertainty, Imperfection,Deliberation* (pp. 1-32). (Studies in Computational Intelligence; Vol. 538). Springer. https://doi.org/10.1007/978-3-319-15144-1_1

Chapter 1
Bayesian Methods for Intelligent Task Assignment in Crowdsourcing Systems

Edwin Simpson, Stephen Roberts

Abstract

In many decision-making scenarios, it is necessary to aggregate information from a number of different agents, be they people, sensors or computer systems. Each agent may have complementary analysis skills or access to different information, and their reliability may vary greatly. An example is using crowdsourcing to employ multiple human workers to perform analytical tasks. This chapter presents a an information-theoretic approach to selecting informative decision-making agents, assigning them to specific tasks and combining their responses using a Bayesian method. For settings in which the agents are paid to undertake tasks, we introduce an automated algorithm for selecting a cohort of agents (workers) to complete informative tasks, hiring new members of the cohort and identifying those members whose services are no longer needed. We demonstrate empirically how our intelligent task assignment approach improves the accuracy of combined decisions while requiring fewer responses from the crowd.

1.1 Introduction

In many scenarios, decisions must be made by combining information from a number of different agents, be they people, sensors or computer systems. These agents may possess useful analytical skills that cannot easily be replicated, or they may have access to complementary information. For example, the fields of crowdsourcing and citizen science often employ human annotators to classify a dataset, since people have sophisticated pattern-recognition and reasoning skills and the ability to learn new tasks given simple, natural language instructions. A large number of

Edwin Simpson
Machine Learning Research Group, University of Oxford, e-mail: edwin@robots.ox.ac.uk,
Stephen Roberts
Machine Learning Research Group, University of Oxford, e-mail: sjrob@robots.ox.ac.uk

annotators can be used to compensate for the limited time that each person can dedicate to the labelling task, and for the use of non-expert and potentially untrusted individuals. Agents may also provide diverse observations for applications such as situation awareness, where information can be obtained from mobile sensors, cameras and human reporters to build an overview of events in a particular scenario. By obtaining and aggregating information from a pool of decision-making agents, we can form a combined decision, such as a classification or an action, taking advantage of the wealth of existing skills, knowledge and abilities of the decision-making agents.

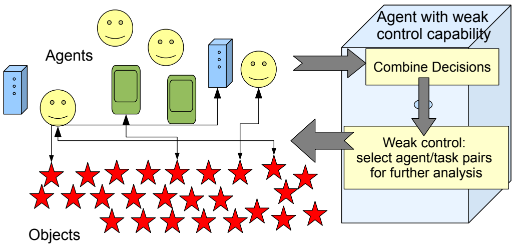
Agents
Agent with weak control capability
Combine Decisions
Weak control: select agent/task pairs for further analysis
Objects

Fig. 1.1 Overview of the intelligent tasking problem: how to assign tasks to agents given current combined decisions.

The canonical situation we consider in this chapter is depicted in Figure 1.1, showing a crowd of agents providing making decisions about a set of *objects*, which can be data points, text documents, images, locations in space and time, or other items about which a decision is required. The right-hand side of the diagram shows an agent that combines decisions from the crowd, then exerts *weak control* to influence the assignment of agents to objects, represented in the diagram by connecting arrows. Weak control consists of suggestions and rewards for completing tasks that meet the weak controller's goals, and is used in situations where the controller cannot force agents to complete particular tasks. Such a situation occurs when working with human agents, who may choose whether to accept or reject tasks, but may be directed toward completing tasks that are informative to the combiner.

Our previous work [22] focused on principled, Bayesian methods for aggregating responses from multiple decision-making agents, and inferring agent reliability. In this chapter we consider the complete system for selecting informative agents, assigning them to specific tasks and combining their responses. Both the choice of task and the suitability of an agent's skills for that particular task affect the utility of the information we can obtain. By deploying agents effectively, we can minimise the number of responses required to confidently learn a set of target decisions. This allows us to analyse larger datasets, reduce the time taken or decrease costs such as payments required by workers in a crowdsourcing system. We therefore propose an information-theoretic approach, *intelligent tasking*, to estimate approximately optimal task assignments, which can exploit additional descriptive information obtained through computational analysis of the objects or environment of interest. For settings in which the agents are paid to undertake tasks, we introduce an automated method for selecting a cohort of agents (workers) to complete informative tasks, hiring new members of the cohort and identifying those members whose services are no longer needed. The results demonstrate clear advantages over more simplistic approaches, but also indicate opportunities for future work, for example to automate agent training and motivate human analysts.

This chapter begins by looking at related work on information aggregation systems and whether they account for these issues. A case study is then introduced for a crowdsourcing system in which it is important to select and deploy agents efficiently. In this scenario, we wish to classify a large dataset given a small subset of unreliable, crowdsourced labels. To do so, we extract features from the objects and use the crowdsourced subset of labels to learn how the features relate to the target classes. To handle the unreliability of the crowdsourced labels, we propose extending a Bayesian approach to decision aggregation, namely *Dynamic Independent Bayesian Classifier Combination (DynIBCC)* [22], to augment discrete agent decisions with continuous object features in the range $[0,1]$. This extension is demonstrated with the crowdsourcing case study, attaining strong performance with limited data. We then introduce an intelligent tasking framework for optimising the deployment of agents, balancing the cost of each task with a desire to maximise information gain. This framework naturally negotiates the need to explore and exploit the agents' skills. The approach is used to develop the *Hiring and Firing* algorithm, which addresses the need to select *both* tasks and agents in a unified manner, and shows promising results in our experiments. The final section of this chapter discusses opportunities for extending intelligent tasking by considering delayed rewards, including those obtained through training and motivation of human agents.

1.2 Related Work

In many existing systems, there is no attempt to select agents to perform particular tasks based on ability or diversity of skills. In Citizen Science applications, such as Galaxy Zoo [25], the aim is to assign more agents to a task until a clear combined decision has been made. For example, Galaxy Zoo Supernovae [24], prioritises objects that have no classifications, and does not request labels for those that already have a sufficient number of answers that agree. The remaining objects are prioritised according to the probability of a positive example. Thus, the system uses a heuristic method to label uncertain objects. The choice of whether to hire more agents to classify a Galaxy Zoo object is addressed by [15] using a partially-observable Markov Decision process (POMDP), but this choice is not tailored to the individual agents, which are not modelled in their approach.

Related work on crowdsourcing has considered the problem of selecting trustworthy workers. Web-based crowdsourcing platforms such as *Amazon Mechanical*

Turk (AMT)[1] allow workers to receive payments for tasks presented through its web interface, but have been shown to suffer from unreliable workers, including spammers who guess random answers to complete tasks more rapidly for money [5; 14]. Some systems focus on rejecting unreliable workers, but assume constant reliability [21; 14; 18]. For example, [21] provides a mechanism for blocking spammers on the fly. In [18], various algorithms are presented for inferring a single reliability metric, where priors can be set to identify workers as spammers or hammers, i.e. trustworthy workers. A method proposed by [14] infers a single error and bias measure per agent for blocking unreliable workers. Since these methods do not model the changing worker dynamics, they in effect treat agents' distant past responses as a significant indication of current reliability. Thus they are unable to account for learning, boredom, or the movement of agents who are mobile observers. Worker dynamics are addressed by [9], who demonstrate how to reject poor performers from a pool of workers by thresholding a changing reliability value.

In other work on crowdsourcing by [20], tasks are allocated to either humans or artificial agents according to speed, cost and quality constraints. However, the system makes decisions according to prior knowledge about agent types rather than observing the abilities of individuals.

The methods discussed above are restricted to iteratively filtering workers, and do not consider the choice of task, e.g. which object to label, which affects the amount of information learned about the target decisions and can influence future behaviour of agents. Most of these methods assign scalar reliability scores [9; 18; 14], so are unable to consider how a worker's reliability varies between types of task, which may be due to their expertise or boredom with a particular type of task. For example, a bored worker may be reinvigorated by completing a different kind of task. Therefore, there are advantages to using a representation of the agents' reliability that accounts for different task types. One such model is a confusion matrix, employed by DynIBCC [22] and related methods [21; 14; 18; 8], in which each row characterises an agent's behaviour with a certain type of task. Each entry in the confusion matrix captures the likelihood of a particular response given the type of task. DynIBCC introduces dynamic confusion matrices that capture variations in reliability over both time and task type. We therefore consider DynIBCC as a model for making combined decisions within the intelligent tasking approach proposed in this chapter.

Several pieces of related work have considered *active learning* with crowdsourcing. Active learning in this context refers to the iterative process of deciding which objects to label, and choosing a labeller, accounting for the potential unreliability of the labeller. In [28], a strategy is developed for binary classification where agents are selected based on how confident they are likely to be for a particular task. However, reliable confidence measures are often unavailable, especially for human agents, and offers no way of handling over-confident or under-confident agents. The work of [6] implements a learning strategy for ranking problems that seeks to maximise expected information gain over both the model and the target variables, introducing

[1] See https://www.mturk.com

a heuristic parameter to balance the exploitation and exploration of the reliability model of the workers. In summary, the related work has not yet produced a principled, unified approach for adaptively selecting individual agents to perform specific decision-making tasks. The remainder of this chapter develops a methodology to account for these factors, beginning in the next section with a case study, to which we our techniques will be applied.

1.3 Case Study: TREC Crowdsourcing Challenge

The work in this chapter relates to an information aggregation problem that requires the efficient use of unreliable workers. As an example of such a problem, we consider the 2012 TREC Crowdsourcing challenge[2], which was a competition to determine whether documents in a given dataset were relevant to a set of 10 search queries. The complete dataset contains 15,424 documents and 18,260 document/query pairs that must be confirmed as true or false. Each search query has a detailed description of a very specific information need, so that it is not possible to confidently judge relevance by searching for a short text string. Examples of topic titles include “definition of creativity” and “recovery of treasure from sunken ships”, with the descriptions that follow specifying the query more precisely. The documents were compiled into the TREC 8 corpus, originally sourced from the *Financial Times*, *Los Angeles Times* and *Federal Register*.

The aim of the challenge was to use crowdsourcing to obtain accurate document relevance classifications, so no training examples were provided for the given queries. However, with a large number of document/query pairs, it is desirable to reduce the number of relevance judgements we need to obtain from the crowd to limit the time and cost taken to classify the complete dataset. Given a subset of crowdsourced training examples, we can use textual features extracted from the documents to predict the labels of documents that have not been processed by the crowd. These predictions could potentially be used to prioritise documents for further crowdsourcing, for example, where their classification is most uncertain. This chapter therefore presents an approach that employs more expensive human agents only when necessary, using cheaper automated techniques when possible, aggregating both types of information.

Bayesian Classifier Combination is an effective approach to aggregating responses from unreliable agents, and has been used in separate applications to combine nominal decisions made by people [22] and to aggregate binary textual features [17], by treating features in the same way as responses from agents. However, textual features may also include unbounded discrete variables, such as word counts, or continuous variables, such as probabilities. It may not be possible to compress all kinds of feature to a small number of categories without losing important information. It is unclear how to choose a threshold for converting a continuous variable to

[2] The Text REtrieval Conference, or TREC, consists of several competitions. For the crowdsourcing challenge, see https://sites.google.com/site/treccrowd/.

a discrete value, so an additional optimisation step may be required to find suitable thresholds. To address this issue, the next section extends the Dynamic Independent Bayesian Classifier Combination (DynIBCC) model [22] to handle continuous features in the range [0,1], thereby enabling the use of DynIBCC for the TREC Crowdsourcing challenge. The following section then presents the results of the competition, and discusses some of the crowdsourcing issues arising in this scenario that motivate a unified intelligent tasking approach.

1.4 DynIBCC for Combining Probabilities

Dynamic Independent Bayesian Classifier Combination (DynIBCC) has been shown to be an effective method for aggregating decisions from multiple agents, while learning how their reliability varies over time [22]. Tracking changing agent behaviour allows DynIBCC to account for human agents who learn or become bored with repetitive crowdsourcing tasks, for example. DynIBCC handles uncertainty in the model and the combined decisions using the principled mathematical framework of Bayesian inference. This section begins by explaining the DynIBCC model and showing how it can be extended to accommodate a mixture of discrete variables and continuous variables in the range $[0,1]$. The following subsection then describes an efficient inference algorithm for DynIBCC, which outputs posterior distributions over a set of target decisions, and over a model of the individual agents' behaviour.

The graphical model for the modified DynIBCC approach is shown in Figure 1.2 and is described as follows. The aim is to infer a set of target decisions $\boldsymbol{t} = \{t_i | i = 1,..,N\}$, given a set of agents' responses $\boldsymbol{c} = \{c_i^{(k)} | i = 1,..,N, k = 1,..,K\}$, and features $\boldsymbol{y} = \{y_i^{(f)} | i = 1,...,N, f = 1,..,F\}$. As with standard DynIBCC proposed in [22], target decisions t_i are drawn from a multinomial distribution with proportions $\boldsymbol{\kappa}$.

The variables in the blue plate on the left relate to the agents $k = 1,..,K$, and are also the same as standard DynIBCC. The likelihood of response $c_i^{(k)}$ from agent k to object i is given by the confusion matrix, $\boldsymbol{\Pi}_\tau^{(k)}$. Each element of this matrix represents their probability of response given an object with a particular target class $t_i = j$, at a time τ:

$$\pi_{\tau,j,l}^{(k)} = p(c_i^{(k)} = l | t_i = j) \tag{1.1}$$

where $l = 1,..,L$ is an index into possible agent responses, $j = 1,..,J$ is an index into the ground truth target decisions and $p(c_i^{(k)} = l)$ represents the probability of a particular value l of the variable $c_i^{(k)}$. The notation $p()$ is used throughout to represent the probability of a variable. The subscript τ is the time-step at which agent k labelled object i, so is a position in a sequence of responses from k. Here, we assume that each response by an agent occurs at a separate discrete time-step, but we could also consider time-steps that represent user sessions or periods in which agents supply multiple responses. The time-dependent confusion matrix $\boldsymbol{\Pi}_\tau^{(k)}$ captures the

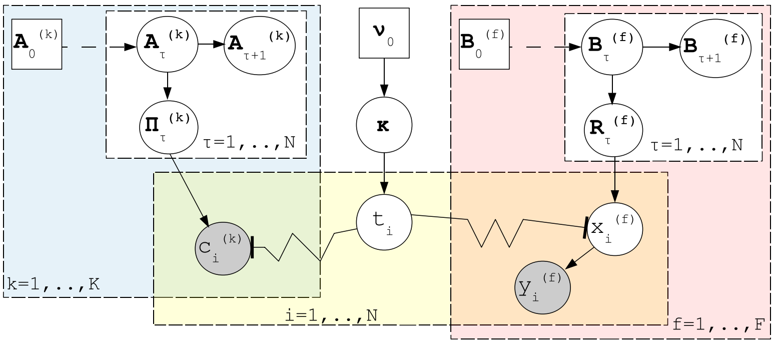

Fig. 1.2 Graphical model for DynIBCC extended to accommodate continuous features. Dashed arrows indicate dependencies on nodes at other time-steps. The zig-zag line means t_i is a switch that selects parameters from $\Pi_\tau^{(k)}$. The shaded node represents observed values. Circular, hollow nodes are variables with a distribution. Square nodes are variables instantiated with point values. The blue, left-hand plate corresponds to agents $k = 1,..,K$, who each produce a discrete response $c_i^{(k)}$ with a probability $\boldsymbol{\Pi}_{\tau,t_i}^{(k)}$, which is selected by the value of a target decision, t_i. Parameter $\boldsymbol{\Pi}_{\tau,t_i}^{(k)}$ is drawn at time-step τ from a Dirichlet distribution with hyperparameters $\boldsymbol{A}_\tau^{(k)}$. The yellow, lower-middle plate corresponds to the target objects $i = 1,..,N$, and the red, right-hand plate corresponds to object features $f = 1,..,F$. Each object feature has a latent discrete value $x_i^{(f)}$, which is drawn with probability $\boldsymbol{R}_{\tau,t_i}^{(f)}$, selected by target value t_i. Parameter $\boldsymbol{R}_{\tau,t_i}^{(f)}$ is drawn at time-step τ from a Dirichlet distribution with hyperparameters $\boldsymbol{B}_\tau^{(f)}$. The target decisions are drawn from class proportions $\boldsymbol{\kappa}$, with hyperparameter‘$\boldsymbol{\nu}_0$.

changing relationship between responses from k and the target decisions we wish to infer. The use of confusion matrices allows us to treat the agents' responses as data, so we can include agents whose responses are correlated with the target decision but are not direct estimates of that decision. We can also include responses that are the opposite of what we expect, and agents whose predictive accuracy varies between different target classes.

The Bayesian treatment of the model requires us to account for uncertainty in model parameters, including $\boldsymbol{\kappa}$ and $\boldsymbol{\pi}_{\tau,j}^{(k)}$, by placing a distribution over their values. Therefore, we assume a Dirichlet distribution for $\boldsymbol{\kappa}$, with hyperparameters $\boldsymbol{\nu}_0$. For the confusion matrices, the jth row $\boldsymbol{\pi}_{\tau,j}^{(k)}$ of $\boldsymbol{\Pi}_\tau^{(k)}$ has a Dirichlet prior whose hyperparameters form a matrix $\boldsymbol{A}_\tau^{(k)}$.

The model is extended from that of through the addition of the right-hand plate, shaded pink, which relates to continuous features. For features $f = 1,..,F$ we observe a probability vector $\boldsymbol{y}_i^{(f)}$ of a latent response variable $x_i^{(f)}$, which may take values $l = 1,..,L$. In effect, we could view these observed probabilities as spreading a single response between multiple discrete values. Each feature, f, has time-dependent confusion matrix, $\boldsymbol{R}_\tau^{(f)} = \{\boldsymbol{\rho}_{\tau,j}^{(f)} | j = 1,..,J\}$, where each row $\boldsymbol{\rho}_{\tau,j}^{(f)}$ is a parameter vector for a categorical distribution with elements

$$\rho_{\tau,j,l}^{(f)} = p(x_i^{(f)} = l | t_i = j, \boldsymbol{R}_\tau^{(f)}, \tau = r(i,f)), \tag{1.2}$$

where $r(i,f)$ maps the object index i to time τ at which the feature f was recorded for i. For many features, it may be appropriate to assume a static confusion matrix. However, dynamics may be important if the target decisions undergo concept drift, or the feature is a sensor reading for a moving target object. The rows in the confusion matrix each have a Dirichlet prior with hyperparameters $\beta_{\tau,j}^{(f)}$. The matrix of hyperparameters for all target values j is referred to as $\boldsymbol{B}_\tau^{(f)} = \{\boldsymbol{\beta}_{\tau,j}^{(f)} | j = 1,..,J\}$.

Features are modelled using confusion matrices in a similar manner to agents, but are separated here for clarity since the discrete value $x_i^{(f)}$ is unobserved. As with the agents' responses, model assumes conditional independence of features given the target labels $\boldsymbol{t}$. The observed vector $\boldsymbol{y}_i^{(f)}$ describes a categorical distribution over the feature value, such that $y_{i,l}^{(f)} = p(x_i^{(f)} = l)$.

To write the complete model we first define some notation. We define the sets of all agents' confusion matrices for all time-steps as $\underline{\boldsymbol{R}} = \{\boldsymbol{R}_\tau^{(f)} | \tau = 1,..,T^{(f)}, f = 1,...,F\}$ and all features' confusion matrices for all time-steps as $\underline{\boldsymbol{\Pi}} = \{\boldsymbol{\Pi}_\tau^{(k)} | \tau = 1,..,T^{(f)}, k = 1,...,K\}$. Since the order in which an agent sees the objects can vary, the time-steps are referenced by the subscript $\tau = s(i,l)$, which is the time-step at which agent k classified object i. The complete model for the extended DynIBCC is represented by the joint distribution:

$$p(\boldsymbol{t}, \boldsymbol{y}, \boldsymbol{c}, \boldsymbol{\kappa}, \underline{\boldsymbol{R}}, \underline{\boldsymbol{\Pi}} | \boldsymbol{B}_0, \boldsymbol{A}_0, \boldsymbol{\nu}_0) = \prod_{i=1}^{N} \left\{ \kappa_{t_i} \prod_{k=1}^{K} \pi_{s(i,k),t_i,c_i^{(k)}}^{(k)} \cdot \prod_{f=1}^{F} \sum_{l=1}^{L} y_i^{(f)} \rho_{r(i,f),t_i,l}^{(f)} \right\} p(\boldsymbol{\kappa} | \boldsymbol{\nu}_0)$$
$$\prod_{\tau=1}^{N} \prod_{j=1}^{J} \left\{ \prod_{k=1}^{K} p\left(\boldsymbol{\pi}_{\tau,j}^{(k)} | \boldsymbol{\alpha}_{\tau,j}^{(k)}\right) p\left(\boldsymbol{\alpha}_{\tau,j}^{(k)} | \boldsymbol{\alpha}_{\tau-1,j}^{(k)}\right) \cdot \prod_{f=1}^{F} p\left(\boldsymbol{\rho}_{\tau,j}^{(f)} | \boldsymbol{\beta}_{\tau,j}^{(f)}\right) p\left(\boldsymbol{\beta}_{\tau,j}^{(f)} | \boldsymbol{\beta}_{\tau-1,j}^{(f)}\right) \right\}, \tag{1.3}$$

where $\boldsymbol{A}_0 = \{\boldsymbol{A}_0^{(k)} | k = 1,..,K\}$ is the set of prior hyperparameters for all agents' confusion matrices at the first time-step, and $\boldsymbol{B}_0 = \{\boldsymbol{B}_0^{(f)} | f = 1,..,F\}$ is the set of prior hyperparameters for all features' confusion matrices at the first step.

1.4.1 Variational Inference Algorithm for DynIBCC

Given the model described above, we require an inference algorithm to evaluate the posterior distribution over the unknown variables, given a set of responses from the crowd $\boldsymbol{c}$ and features of the objects $\boldsymbol{y}$. The goal of inference is to estimate the unknown target decisions $\boldsymbol{t}$ and model parameters $\underline{\boldsymbol{\Pi}}$, $\underline{\boldsymbol{R}}$, and $\boldsymbol{\kappa}$. A principled yet efficient inference algorithm can be derived using variational Bayesian (VB) inference and is described in detail in [22]. This method obtains distributions over the unknown variables in either an unsupervised or semi-supervised manner, naturally exploiting any training examples of t_i that are available. We initialise the algorithm by setting starting values for expectations with respect to the target decisions and model parameters. The algorithm then operates in an iterative manner, alternating between two steps until convergence:

1. Update approximate posterior distributions over the model parameters, $\underline{\boldsymbol{\Pi}}$, $\underline{\boldsymbol{R}}$ and $\boldsymbol{\kappa}$, given the current estimates of expected values of the target decisions $\boldsymbol{t}$. For any known training examples, use the true values rather than expectations.
2. Update approximate posterior distributions over the unknown target decisions, $\boldsymbol{t}$, given the current estimates of expectations with respect to the model parameters, $\underline{\boldsymbol{\Pi}}$, $\underline{\boldsymbol{R}}$ and $\boldsymbol{\kappa}$.

In step 1, the computational cost is dominated by the total number of responses all agents, $N_{\text{responses}})$, and the amount of feature data. Updating terms relating to $\underline{\boldsymbol{\Pi}}$ therefore has complexity $\mathscr{O}(N_{\text{responses}}J)$, where J is the number of target classes. Updating the terms relating to $\underline{\boldsymbol{R}}$ has complexity $\mathscr{O}(NFJ)$, where N is the number of objects and F is the number of features. The second step scales similarly with $\mathscr{O}(N_{\text{responses}}J + NFJ)$. Thus the algorithm's costs can be seen as growing linearly with the amount of data and number of possible target class values. Much of the updating can be performed in parallel, since the updates corresponding to each target decision t_i are independent of the updates for other objects i, and each series of confusion matrices $\boldsymbol{\Pi}^{(k)}$ or $\boldsymbol{R}^{(f)}$ is similarly updated independently of the others.

The number of iterations required by VB depends on the initial values used, since if these values are close to the final estimates, few iterations will be required. Therefore if we receive a new observation from the crowd, we can perform an efficient update by restarting the iterative process from our previous estimates, assuming that a single observation will change the posterior distribution only a small amount. In [22], the VB algorithm was compared using real crowdsourcing data to another Bayesian approach, Gibbs' sampling, which is a Markov-Chain Monte Carlo method [11]. When applied to a sample of this dataset containing 10,000 crowdsourced responses and 660 target objects, the VB algorithm required at most 10 iterations and 0.4 seconds to converge on a standard desktop workstation, while performing Bayesian inference using required 170 iterations and 3.33 seconds to obtain the same accuracy. At the same time, the Gibbs' sampling algorithm did not substantially increase the accuracy of results compared to the VB algorithm. Therefore the VB algorithm provides an accurate, scalable, fully-Bayesian method for updating our posterior distribution online as new labels are received from a crowd.

This extended variant of DynIBCC allows us to obtain agent responses for only a subset of objects, but predict target decisions for all objects, including those that have not been labelled directly by the crowd. In situations such as the TREC challenge, we have descriptive features for all objects, such as the words in the text, so we can use this method to learn the feature confusion matrices $\underline{\boldsymbol{R}}$. The Bayesian approach quantifies the uncertainty in the feature confusion matrices, along with the other parameters and latent variables in the model. As the next sections will show, this allows us to evaluate the utility of obtaining additional labels from agents to reduce this uncertainty and increase our confidence in the target decisions.

1.5 TREC Results using DynIBCC + Text Features

The TREC crowdsourcing challenge was addressed through the novel application of the extended DynIBCC approach. This method allowed the entire corpus of $15,424$ documents to be classified with no *a priori* training labels, by combining $2,500$ crowdsourced labels (16% of the corpus) with 2000 textual features for each document. crowdsourced labels were supplied to DynIBCC from Amazon Mechanical Turk (AMT), where human agents completed tasks involving reading a document, then determining a label, which was either one of ten search queries or the option “none of the above”. Further details of the crowdsourcing system are described in [23].

The system also provided textual features from the documents using Latent Dirichlet Allocation (LDA) [4]. LDA infers a distribution over topics for each document according to the words it contains, so that in this implementation, each document is associated with a vector of 2000 probability values. These probability values are treated as observations $\boldsymbol{y}$ of feature values, which are combined with the crowdsourced responses, $\boldsymbol{c}$, using the Bayesian Classifier Combination method described in the previous section, referred to here as simply *DynIBCC*.

The system was evaluated by examining 18,260 document/query pairs, which were verified by a committee as true or false matches [26]. Using the same set of crowdsourced labels, DynIBCC was compared to a two-stage naïve Bayes method [23], referred to here as *2StageNB*. For this experiment, the confusion matrices in DynIBCC were fixed so that they did not vary over time. The two-stage method used a training phase to learn likelihood distributions for binary features given each target class, treating the crowdsourced labels as reliable classifications. Unlabelled documents were ignored during the training step and priors were not placed over the model parameters. In the prediction phase, the two-stage method uses the feature likelihood distributions to predict the correct search queries for all documents in the corpus. The results of DynIBCC and the two-stage method were also compared to the systems used by other competitors, which obtained different sets of crowdsourced responses using a variety of approaches.

The results are given in [26] and summarised in Table 1.1 by the area under the receiver operating characteristic curve (AUC). The AUC can be seen as an overall measure of classification efficacy that is independent of the frequency of each class. The AUC gives the probability that the classifier will assign a randomly chosen positive example a higher probability of being positive than a randomly chosen negative example. The AUC is calculated from the receiver operating characteristic (ROC) [10], which is a plot of a classifier’s true positive rate against false positive rate for different values of a threshold used to convert the classifier’s predictions to discrete class values. Each point on the ROC curve therefore corresponds to a particular threshold value. All of the classifier's predictions below that value are taken as negative classifications, and all those above are assumed to be positive. The true positive rate at a particular threshold is the fraction of positive examples correctly identified by the classifier, while the false positive rate is the fraction of negative candidates incorrectly classified as positive. Thus the ROC curve and AUC account for the possibility that we may wish to vary the threshold to optimise the false positive rate and true positive rate for a particular application. The original publication of

Method	No. Labels Collected	Mean AUC	Described in
DynIBCC	2500	0.806	The current section
2StageNB	2500	0.774	The current section
SSEC3inclML	30312	0.914	[19]
UIowaS02r	3520 from crowd + 129 sets of past results	0.881	[12]
NEUNugget12	N/A	0.748	[1]
BUPTPRISZHS	54780	0.597	[29]
INFLB2012	N/A	0.517	N/A
yorku12cs03	N/A	0.479	[13]

Table 1.1 Area under receiver operating characteristic curve (AUC) for competitors in the TREC Crowdsourcing challenge. DynIBCC refers to the method proposed in this chapter, while the simpler two-stage method is 2StageNB. The other method names refer to systems developed by TREC competitors; the important competitors are described in the text.

results [27] did not evaluate the AUCs for runs labelled *UIowaS02r*, *BUTPRISZHS*, *INFLB2012*, and *yorku12cs03*, as these methods produced only binary classifications.

In comparison with the two-stage aggregator, the results show the superior performance of DynIBCC. A likely cause of this increased performance is that DynIBCC accounts for unreliability in the confusion matrices and the crowdsourced labels. In contrast, the two-stage classifier trains the model by assuming these labels are correct and makes predictions assuming that all confusion matrices have been confidently learned.

Both DynIBCC and 2StageNB outperformed several other approaches, although various elements of the crowdsourcing system may have contributed to the system's overall performance. None of the other competitors used a Bayesian decision combination method to account for uncertainty in model parameters relating to the crowd's responses or textual features.

Two competitors – SSEC3inclML and UIowaS02r – outperformed DynIBCC by using a substantially larger amount of data. No limit was placed on the budget allowed for the competition, nor on the number of labels the crowd could provide. SSEC3inclML [19] labelled every document at least once, obtaining a total of $30,312$ labels. Their intention was to obtain reliable labels by using an expert information analyst to train an in-house crowd. Machine Learning techniques analysed text features to flag up possible errors after all documents had been labelled once, so that those documents could be re-labelled. UIowaS02r [12], exploited relevance judgements submitted to a previous competition for the same documents and queries. First, the system ranked the documents in an estimated order of relevance by combining the rankings from 129 earlier submissions. Then, for each query, the 10 highest ranked documents were marked as positive examples for those queries. The remaining documents were labelled iteratively in batches of 20 using crowdsourcing, in order of increasing rank. Once an entire batch had been marked irrelevant, no more batches were sent to the crowd for that search query. While $3,520$ labels were extracted from the crowd, which is approximately 40% more than for DynIBCC, a far larger number of relevance judgements were contained in the data used from the earlier competition (the exact number is not given).

The superior outcomes of SSEC3inclML and UIowaS02r may stem primarily from the far larger numbers of relevance judgements used. However, training the crowd was also a key feature in SSEC3inclML, and both methods focused on labelling difficult or uncertain documents. The information learnt by DynIBCC could be used to select particular documents for crowdsourcing or automatically train unreliable agents, since DynIBCC computes confidence in the target labels $\boldsymbol{t}$ and feature confusion matrices $\underline{\boldsymbol{R}}$, and models the reliability of agents through the confusion matrices, $\underline{\boldsymbol{\Pi}}$. This would require DynIBCC to be run as new labels are received from the crowd.

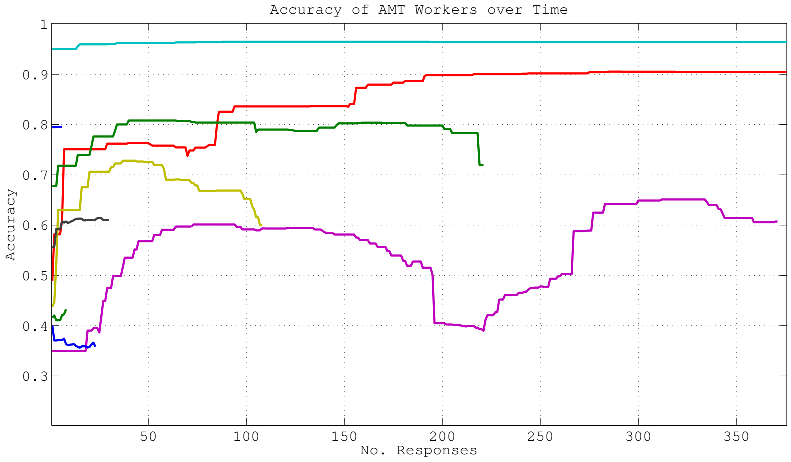

Fig. 1.3 Accuracy of 16 Amazon Mechanical Turk human agents over time, as inferred by DynIBCC-VB. The agents all completed at least 7 out of 10 screening tests correctly. However, note the variation in abilities between agents that is inferred over a longer period. Also note the different changes in accuracy over time.

Since the error rates of the agents affect the accuracy of combined decisions, we performed additional post-hoc analysis using DynIBCC to examine the reliability of the agents. We ran DynIBCC over the final set of crowdsourced responses, given the correct decisions, assuming that the confusion matrices vary over time according to the dynamic model described in [23]. The confusion matrices were then summarised by a single accuracy value a_τ at each time-step τ, calculated as

$$a_\tau = \sum_{j=1}^{J} \left\{ \frac{\mathbb{E}[\pi_{\tau,j,j}^{(k)}]}{\sum_{l=1,l\neq j}^{L} \mathbb{E}[\pi_{\tau,j,l}^{(k)}]} \mathbb{E}[\kappa_j] \right\}, \tag{1.4}$$

where $\mathbb{E}[.]$ is an expected value, as estimated by the variational Bayes inference algorithm for DynIBCC (Section 1.4). Figure 1.3 plots the accuracy a_τ over time, showing significant variation and changes to agents. The system used to crowdsource labels for DynIBCC and 2StageNB employed a simple screening step, in which agents completed ten documents, for which the correct labels were known. Agents were then employed if their accuracy on the test tasks was greater than 0.67. Agents were initialised with the same values for $\boldsymbol{A}_0^{(k)}$ to give an expected accuracy of $a_0 = 0.67$. However, the post-hoc analysis inferred accuracies ranging from approximately 0.35 to 0.96. While some agents appear to have improved over time, there are also those that deteriorate, four of which do so before they stop providing more labels. This perhaps suggests a loss of interest in the highly repetitive tasks, although a thorough investigation of AMT agent behaviour is required to determine the causes of behavioural changes.

The large variation in agent reliability shown in Figure 1.3, suggests that intelligent selection of agents is important, particularly when the budget or time is limited. The communities found within a large citizen science application (see [22]) previously demonstrated the large variation in agents' behaviour in a different context, while related work described in Section 1.2 also identifies problems with spammers in AMT. The varying accuracies shown in Figure 1.3 point to the need for on-going agent selection to maintain an effective pool of agents.

The remainder of the chapter therefore focuses on a theoretically-motivated intelligent tasking approach for agent selection and task assignment. Such a method should be able to make effective decisions when only a small dataset is available, as is the case at the start of the crowdsourcing process when few labels have been received from the crowd.

1.6 A Utility Function for Intelligent Tasking

Intelligent tasking is an information-theoretic approach to determining the optimal action when aggregating information in a multi-agent system. The remainder of this chapter focuses on two key problems that intelligent tasking can solve: (1) selecting informative analysis tasks for agents that enable a model such as DynIBCC to learn the target decisions confidently with minimal crowdsourced labels; (2) selecting and maintaining a reliable pool of agents who can be assigned to analysis tasks and provide informative responses. The core idea is that every action, such as an agent responding to an object, can be evaluated by a utility function that defines value in terms of information gain. Each action consists of an agent, k, performing a task, i, depending on the application. In a citizen science scenario such as Galaxy Zoo Supernovae [24], task indexes typically correspond to data points or images that must be analysed. In scenarios involving mobile agents, tasks may also include moving to observe from a particular location. Besides such information-gathering tasks, agents can also take actions that may lead to rewards in the future, such as carrying out training exercises. It is assumed that the overall goal of the information gathering exercise is to learn the values of a set of target variables, $\boldsymbol{t}$ and that each action generates a new observation, $c_i^{(k)}$. We can define a utility function for the result of an agent k performing task i given previous responses $\boldsymbol{c}$ and object features $\boldsymbol{y}$:

$$U(k,i|\boldsymbol{c}) = \sum_{\iota=1}^{N} I(t_\iota; c_i^{(k)}|\boldsymbol{c},\boldsymbol{y}), \tag{1.5}$$

where $I()$ refers to the Kullback-Leibler *Information Gain* [16]. Kullback-Leibler Information Gain is a suitable choice for defining our utility function because it quantifies the amount of information learned about the target decision t_ι if we can predict t_ι using $p(t_\iota|c_i^{(k)},\boldsymbol{c},\boldsymbol{y})$ rather than $p(t_\iota|\boldsymbol{c},\boldsymbol{y})$. It is defined as:

$$I(t_\iota; c_i^{(k)}|\boldsymbol{c},\boldsymbol{y}) = \sum_{j=1}^{J} p(t_\iota = j|c_i^{(k)},\boldsymbol{c},\boldsymbol{y}) \ln\left(\frac{p(t_\iota = j|c_i^{(k)},\boldsymbol{c},\boldsymbol{y})}{p(t_\iota = j|\boldsymbol{c},\boldsymbol{y})}\right). \tag{1.6}$$

If the logarithms used in this equation are base e, the information gain is measured in nats, and if base 2 is used, the units are bits. Hence we can measure the information obtained from an agent and compare this quantity to the information provided by other agents and by other labelling tasks. Kullback-Leibler Information Gain also depends on the terms in the conditions $\boldsymbol{c},\boldsymbol{y}$, i.e. what we already know, so that information is only valued if it is complementary to what we have already learned.

The true value of the utility function can only become known once we observe the value of $c_i^{(k)}$. *Intelligent tasking* therefore refers to any algorithm that directs agents to tasks that maximise the *expected* utility:

$$
\begin{aligned}
\hat{U}(k,i|\boldsymbol{c}) &= \sum_{\iota=1}^{N} \mathbb{E}_{c_i^{(k)}}\left[I(t_\iota; c_i^{(k)}|\boldsymbol{c},\boldsymbol{y})\right] \\
&= \sum_{\iota=1}^{N} H(t_\iota|\boldsymbol{c},\boldsymbol{y}) - \sum_{c_i^{(k)}=1}^{L} p(c_i^{(k)}=l|\boldsymbol{c},\boldsymbol{y}) H(t_\iota|\boldsymbol{c},\boldsymbol{y},c_i^{(k)}=l) \\
&= \sum_{\iota=1}^{N} \left\{ \sum_{j=1}^{J} \sum_{c_i^{(k)}=1}^{L} p(t_\iota=j, c_i^{(k)}=l|\boldsymbol{c},\boldsymbol{y}) \ln p(t_\iota=j|\boldsymbol{c},\boldsymbol{y},c_i^{(k)}=l) \right. \\
&\qquad \left. - \sum_{j=1}^{J} p(t_\iota=j|\boldsymbol{c},\boldsymbol{y}) \ln p(t_\iota=j|\boldsymbol{c},\boldsymbol{y}) \right\},
\end{aligned}
\tag{1.7}
$$

where $H(x)$ is the Shannon entropy, which evaluates the uncertainty of a random variable x by giving the amount of information learned about x by observing its value. When logarithms are taken to base e, the entropy is measured in nats. The expected information gain is therefore the expectation with respect to the label $c_i^{(k)}$ of the reduction in entropy of the target decision t_ι. This expected information gain can also be referred to as the *mutual information* between a response $c_i^{(k)}$ and a target decision t_ι.

If we take the decision that maximises the expected utility $\hat{U}(k,i|\boldsymbol{c})$, we are also minimising a loss function that is the negative of the utility function. The Bayesian decision rule means we take the decision that minimises the expected loss, which is an *admissible* decision rule, i.e. there is no better decision given our loss function and current knowledge [2]. Thus, given that we have defined utility according to Equation (1.5), the optimal action is to choose the agent-task pair that maximises Equation (1.7). However, Equation (1.5) is a *greedy* utility function, i.e. one that considers only the immediate utility of the next action taken by an agent. If we use the greedy utility function to assign agents to tasks iteratively as each task is completed, we are operating with a greedy strategy. Such a greedy strategy is sub-optimal, meaning that it may not result in the maximum reduction in uncertainty in the target decisions over multiple iterations. This sub-optimality occurs because the utility function does not consider how the immediate response will affect later decisions, nor how future observations might affect the current choice. Therefore, using the greedy strategy to select objects for an agent to analyse will optimise the only the utility of the current assignment, rather than future assignments. However, it leads to far more scalable algorithms and has been shown to be approximately as good as the optimal algorithm for minimising the number of labels required in an *Active Learning* scenario [7]. In applications such as citizen science, it may be necessary to propose several tasks for an agent, since control over the agents is limited to the ability to make suggestions, which may be rejected.

The terms in Equation (1.7) can be obtained by learning the DynIBCC model, or indeed any other Bayesian decision combination model. It is important to use a model that accounts for uncertainty in the model parameters, otherwise we will underestimate the entropy in the target decisions, $H(t_\iota|\boldsymbol{c},\boldsymbol{y})$, so that the information gain predictions are not meaningful. The term $p(t_\iota = j|\boldsymbol{c}, \boldsymbol{y})$ can be estimated by running the DynIBCC algorithm with the current set of observations. The terms $p(t_\iota = j|\boldsymbol{c}, \boldsymbol{y}, c_i^{(k)} = l)$ are calculated by re-running the algorithm for each possible value of $c_i^{(k)} = l$, which is added as a simulated observation to the current observations. To encourage rapid convergence of the estimates for $p(t_\iota = j|\boldsymbol{c}, \boldsymbol{y}, c_i^{(k)} = l)$, we can initialise all variables relating to model parameters $\underline{\boldsymbol{\Pi}}$, $\underline{\boldsymbol{R}}$ and $\boldsymbol{\kappa}$ to their final values from the earlier run of the algorithm used to estimate $p(t_\iota = j|\boldsymbol{c}, \boldsymbol{y})$. If the addition of a single crowdsourced label $c_i^{(k)}$ causes only small changes to the distribution over the target decisions $\boldsymbol{t}$, then the algorithm in Section 1.4 will require very few iterations to converge. Section 1.7 explains further how we can use Equation (1.7) to develop a practical method for selecting agents and analysis tasks in a crowdsourcing application.

1.6.1 Exploitation and Exploration

This section considers some important properties of Equation (1.7), which defines the expected utility of obtaining a label $c_i^{(k)}$ from agent k for target decision i. This utility function naturally balances the value of both *exploitation* and *exploration* of the model. Exploitation refers to using the current model to to learn target decisions $\boldsymbol{t}$ from new crowdsourced responses, while exploration means learning the model itself. When using DynIBCC, exploration involves learning the confusion matrices $\underline{\boldsymbol{\Pi}}$ that describe the agents' behaviour. Exploration is needed to produce a good model that we can then exploit to determine which agents are most informative, and which objects they should analyse. It is therefore important for a utility function to trade-off between these two cases, which Equation (1.7) achieves without including separate terms for exploration and exploitation. To see how Equation (1.7) balances exploitation and exploration, we consider as follows two scenarios where we take a new label $c_i^{(k)}$ for object i.

As the confidence in our model increases, our estimate of the agent's reliability becomes more certain and the entropy over the confusion matrix $H(\boldsymbol{\pi}_j^{(k)}) \rightarrow 0$. With a confident model, we place more value on exploitation, so we expect to see higher utility for an object i where the target decision t_i is uncertain, because we expect the biggest change between $H(t_i|\boldsymbol{c}, \boldsymbol{y})$ and $H(t_i|\boldsymbol{c}, \boldsymbol{y}, c_i^{(k)} = l)$ when initial entropy $H(t_i|\boldsymbol{c}, \boldsymbol{y})$ is high. In contrast, the value of exploration decreases because we can learn little about the model from $c_i^{(k)}$, so for another object ι that agent k does not supply a new response to, there is little difference between $H(t_\iota|\boldsymbol{c}, \boldsymbol{y})$ and $H(t_\iota|\boldsymbol{c}, \boldsymbol{y}, c_i^{(k)} = l)$. A numerical example of the agent in this scenario is shown in Tables 1.2 to 1.4. The agent $k = 1$ in Table 1.3 has highest utility in Table 1.4 if we exploit the model by assigning agent 1 to the uncertain object $i = 1$.

As confidence in our model decreases, the uncertainty over the agent's reliability increases, so the entropy $H(\boldsymbol{\pi}_j^{(k)}) \rightarrow \infty$. If we do not have a certain model of how the response from agent k relates to the target decision t_i, the amount we can learn about t_i from agent k is low. Thus the entropy $H(t_i|\boldsymbol{c},\boldsymbol{y},c_i^{(k)}=l)$ of t_i on observing $c_i^{(k)}=l$ would change little from the initial entropy $H(t_i|\boldsymbol{c},\boldsymbol{y})$, and hence Equation (1.7) would include a low utility for exploiting the model of agent k to learn t_i. However, if the target decision t_i is already known with high confidence, observing the agent's response to object i informs us about their confusion matrix $\boldsymbol{\pi}_j^{(k)}$ and reduces the entropy $H(\boldsymbol{\pi}_j^{(k)})$. This can cause an expected reduction in entropy for an object ι previously classified by agent k, so that the initial entropy $H(t_\iota|\boldsymbol{c},\boldsymbol{y})$ is higher than the expected entropy given $c_i^{(k)}$, which is $\sum_{c_i^{(k)}=1}^{L} p(c_i^{(k)}=l|\boldsymbol{c},\boldsymbol{y})H(t_\iota|\boldsymbol{c},\boldsymbol{y},c_i^{(k)}=l)$. Thus the utility function includes the value of exploring the model through learning about objects that were previously analysed by agent k. The target objects that allow us to explore a model can be *gold-labelled tasks*, where the ground truth is assumed to be known with certainty, or *silver-labelled tasks*), where our model has inferred the target decision with high confidence from previous crowdsourced responses. Table 1.3 shows an uncertain agent $k=2$, which has highest expected utility in Table 1.4 if it analyses object $i=3$, for which the target decision is known with high certainty and is an example of a silver task. As an alternative to approaches that insert gold tasks, silver tasking avoids the need to obtain expert labels that can be treated as ground truth. By using expected information gain, Equation (1.7) provides an automatic method of selecting silver tasks when appropriate, allowing us to explore a model in a completely unsupervised situation. By defining the expected utility in

Obj. ID	$i=1$ Unlabelled, uncertain	$i=2$ Incorrectly labelled	$i=3$ High certainty
Entropy in target decision, $H(t_i\|\boldsymbol{c},\boldsymbol{y})$	2.3	2.3	0.5448

Table 1.2 Numerical example of utility: uncertainty in target decisions for a set of example objects, measured using Shannon entropy H (nats).

Agent ID	$k=1$ Reliable, certain confusion matrix	$k=2$ Uncertain confusion matrix	$k=3$ Unreliable, certain confusion matrix
Entropy in confusion matrix, $H(\boldsymbol{\pi}^{(k)}\|\boldsymbol{c},\boldsymbol{y})$	0.6	12.8	1.2

Table 1.3 Numerical example of utility: uncertainty in agents' confusion matrices, measured using Shannon entropy H (nats).

	Object ID		
	$i = 1$	$i = 2$	$i = 3$
Agent ID: $k = 1$	20	10	≈ 0
$k = 2$	≈ 0	≈ 0	4.7
$k = 3$	≈ 0	≈ 0	≈ 0

Table 1.4 Numerical example of utility: the expected utility of each agent analysing each object, given by the expected information gain for pairs of agents k and objects i. Note how $k = 1$ has a well-known confusion matrix so can be exploited to label an uncertain document, while $k = 2$ has an uncertain confusion matrix that must be explored by labelling a well-known document. The unreliable agent has low value in either case, so can be fired in exchange for a new agent.

Equation (1.7) as a sum of expected information gain for all target decisions, we avoid the need for any explicit exploitation/exploration parameters, as the balance arises naturally. The next section develops an intelligent tasking algorithm employs our definition of expected utility to simultaneously maintain a reliable workforce while selecting informative analysis tasks.

1.7 Hiring and Firing for Crowdsourcing

This section develops an intelligent tasking algorithm suitable for task and agent selection in the TREC crowdsourcing scenario. The approach lays the foundations for more comprehensive intelligent tasking algorithms that tackle additional decision-making problems in multi-agent systems, such as training and motivating people. As a basis, the aim is to select task assignments that maximise the expected utility given in Equation (1.7), in order to learn a set of target decisions with confidence using a minimal number of crowdsourced responses. This section begins by outlining a number of assumptions that allow us to develop a tractable intelligent tasking algorithm for applications such as the crowdsourcing case study considered earlier in this chapter.

The first assumption is that multiple tasks can be carried out concurrently by different agents. While it may seem preferable to use only the best agent available, in practice this agent is unknown and it is desirable to use several agents to obtain responses more quickly to meet time constraints. When few gold labels are available, observing multiple agents improves the model's ability to distinguish reliable agents, since agents that agree are less likely to be guessing answers at random. The algorithm proposed in this section therefore assumes a fixed pool size, N_{poolsize}, which is the number of agents currently employed.

The second assumption is that after an agent completes a task, they can either be re-hired immediately or fired permanently. This arises because if there is a delay in presenting new tasks, agents on platforms such as Amazon Mechanical Turk (AMT) are likely to find an alternative application to work on, so cannot be re-hired after a period of no tasks.

The final assumption is that new agents are always available to be hired to replace fired agents or those that choose to leave, and that there is no penalty when replacing agents. This assumption is suited to large crowdsourcing platforms such as AMT, where a very large number of agents are easily accessible at any time.

Given the above assumptions, we can specify an algorithm that maintains a pool of trustworthy agents by replacing those who are uninformative with new agents. The algorithm should aim to make the optimal decision each time a new response from an agent is observed: either hire the agent to perform the optimal task for that agent, or fire the agent and hire a new agent to perform the optimal task for a new agent. To make this decision, the algorithm evaluates Equation (1.7) multiple times to determine the expected utility of assigning different objects to the current agent and to a new, *unknown agent* from whom we have not yet observed any responses. In the case that the unknown agent has higher expected utility, they are hired to replace the current agent. The expected utility $\hat{U}(u,i|\mathbf{c},\mathbf{y})$ of the unknown agent, u, depends on the prior distributions over each row of the confusion matrix $\boldsymbol{\pi}_j^{(u)}$. Informative priors can be set by observing the performance of agents in the same crowdsourcing system on a previous set of documents, and taking a mean of their response counts. The magnitude of the counts must then be reduced so that the variance of $\boldsymbol{\pi}_j^{(u)}$ matches the sample variance of the observed agents.

1.7.1 Hiring and Firing Algorithm

The *Hiring and Firing* algorithm for intelligent task assignment operates according to the following steps:

1. Initialise the set of hired agents, $\boldsymbol{H} = \emptyset$, and idle hired agents, $\boldsymbol{H}_{\text{idle}} = \emptyset$.
2. Run DynIBCC over current set of data, $\{\boldsymbol{c},\boldsymbol{y}\}$ to obtain probabilities of labels for all objects. Initially, crowd responses $\boldsymbol{c}$ are empty, and we only see features $\boldsymbol{y}$.
3. Calculate $\hat{U}(k,i|\mathbf{c},\mathbf{y})$ for all tasks, i, and all available agents, $k \in \boldsymbol{H}_{\text{idle}}$ and for an unknown new agent, u.
4. Set $N_{\text{toassign}} = N_{\text{poolsize}} - N_{\text{hired}} + N_{\text{idle}}$ where N_{toassign} is the number of agents to assign, and N_{poolsize} is the desired agent pool size, N_{hired} is the number of agents we currently have in our pool, and N_{idle} is the number of agents in our pool that are not currently working on a task. The number to assign is therefore the shortfall in the current agent pool plus the number of idle agents.
5. While $N_{\text{toassign}} > 0$:

   a. Where $k \in \{W_{\text{idle}}, u\}$ is any available agent, including an unknown agent, u, choose the assignment (k,i) that maximises the expected utility, $(k,i) = \underset{k,i}{\operatorname{argmax}}\ \hat{U}(k,i|\mathbf{c},\mathbf{y})$. The chosen agent is *hired* to do task i. Do not consider any tasks that are currently being completed by other agents, as the other responses are likely to significantly reduce the utility of any repeated assignments.

b. Remove i from the list of possible task assignments for this iteration to avoid repeating the task.

c. If k is not the unknown agent, remove k from W_{idle} as they are no longer idle.

d. Set $N_{\text{toassign}} = N_{\text{toassign}} - 1$.

6. Any agents remaining in $\boldsymbol{H}_{\text{idle}}$ are *fired* and removed from $\boldsymbol{H}$ and $\boldsymbol{H}_{\text{idle}}$
7. Send the selected task/agent pairs to the crowdsourcing system for agents to complete in parallel; await responses.

   a. Any agents who have not yet been hired or fired can complete tasks assigned to u. When an unknown agent accepts a task, they are added to the pool, $\boldsymbol{H}$, and are no longer treated as unknown.
   b. A time-out occurs if a task is not completed within a specified period. The assigned agent is then removed from $\boldsymbol{H}$, and the process is repeated from Step 2 to hire a replacement agent.
   c. On receiving a new label from agent k, add k to $\boldsymbol{H}_{\text{idle}}$, then repeat from Step 2.

In Step 7c, the first iteration will result in $\boldsymbol{H}_{\text{idle}}$ containing only the first agent to complete their task. In subsequent iterations, multiple agents could have been added to $\boldsymbol{H}_{\text{idle}}$ while the other steps of the algorithm were being computed. A delay in Step 7c before repeating could be added to wait for more idle agents before the algorithm is repeated, but the delay should not be long enough to dissuade agents from completing more tasks. Alternatively, multiple iterations of the algorithm could run in parallel as each agent completes a task, so that $\boldsymbol{H}_{\text{idle}}$ typically contains only one agent. Assuming that each assignment is made using all labels currently available from agents, and the best agent/task pair is chosen from those currently available, the algorithm is the locally-optimal *greedy* approach. That is, we cannot improve the expected utility of the next assignment by using any other decision rule. This process combines the screening of agents with selection of informative tasks, avoiding the need for a separate method to test and screen agents periodically.

1.7.2 Computational Complexity

In the hiring and firing algorithm presented above, step 3 is the most computationally expensive part as it requires us to calculate the utility $\hat{U}(k,i|\mathbf{c},\mathbf{y})$ for all available workers and tasks. In the worst case this requires $\mathscr{O}(N_{\text{poolsize}}NJL)$ calculations of $\hat{U}(k,i|\mathbf{c},\mathbf{y})$, where we have N_{poolsize} hired workers, N possible analysis tasks, J target class values and L possible response values from the crowd. Within each of these calculations we must update the DynIBCC model using the VB algorithm. The number of iterations required is typically small even for large datasets, and the calculations required for each iteration scale with $\mathscr{O}(N_{\text{responses}}J + NFJ)$, where $N_{\text{responses}}$ is the total number of responses received from the crowd.

When dealing with large numbers of objects, the computational cost of Step 3 can be reduced by considering only a subset of objects. This allows us to reduce the $\mathscr{O}(N_{\text{poolsize}}N)$ term to $\mathscr{O}(N_{\text{poolsize}}N_{\text{subset}})$, where N_{subset} is a small constant and no longer grows if we have more objects to analyse. For a fair comparison between agents, the same subset should be used for all agents in one iteration. The aim is to use a computationally less costly method to obtain a subset of tasks that contains at least one with expected utility, $\hat{U}(k,i|\mathbf{c},\mathbf{y})$, close to that of the locally-optimal task. In theory, we can improve over selecting tasks for agents entirely at random by simply using the hiring and firing approach to choose objects from a random subset.

It is possible to select a subset of candidate tasks in an informed way by considering relationships between objects with similar features and crowd responses. Objects with similar features are likely to have similar utility, since they will have similar posterior distributions over their target decisions, and a new crowdsourced response would affect those posteriors by a similar amount. Therefore, by sampling the utility function at different points in feature space, we can search for objects that are close to the locally-optimal expected utility.

To explore the whole feature space, we can obtain a representative sample by first grouping similar objects into clusters, then selecting a representative object from each cluster. In the case of the TREC document corpus, each cluster would represent documents with similar word features and similar topics. If we use small, highly-specific clusters, we are more likely to sample points that are close to the optimum, but the size of N_{subset} will increase. It may also be possible to optimise our choice of task by repeatedly selecting and evaluating subsets of documents, thus improving expected utility over that of the initial subset.

The following experiments show that we can obtain good performance by running a clustering algorithm once for each iteration of hiring and firing before step 3. We then choose a representative object from each cluster at random. This allows us to fix the size of subset N_{subset} for which the hiring and firing algorithm must evaluate expected utility. The experiments below use K-means clustering [3] with $k = N/25$. This approach explores the feature space while avoiding redundant comparisons of highly similar object-agent pairs. Hence, we can limit computational cost while selecting tasks that provide higher expected information gain than random selection.

1.8 Hiring and Firing Experiments

The Hiring and Firing algorithm is compared with four other methods described below using simulated agents on 600 documents from the TREC crowdsourcing dataset. Of the documents selected, 37 belong to topic 427 from the TREC8 dataset, while the rest are randomly selected from documents that were not marked as relevant to the above topic. This experiment combines the same LDA features used in

Section 1.5 with the simulated agent responses. The experiment was repeated over 20 datasets, each including different irrelevant documents; each algorithm was run once over all of the datasets.

1.8.1 Simulated Agents

This experiment used simulated agents so that equivalent behaviour can be replicated for each of the algorithms tested. As with the TREC crowdsourcing scenario, agents are assigned documents by a centralised decision maker, and label them as relevant to topic 427 or not relevant. The agents' responses are drawn from a categorical distribution with a predefined probability of drawing the true category for a document. As new agents are hired, the probability of drawing the true category is initialised to either 0.95, 0.8, or 0.5, chosen at random. The initial accuracy cycles through these values as new agents are generated. Thus the hired agents have mixed reliability from very accurate to uninformative. The ideal performance of the algorithms is to fire all but the most reliable agents.

To test the ability of the algorithms to deal with deterioration in behaviour, the agents switch abruptly to an uninformative mode after between 10 and 25 iterations. In the uninformative mode, the correct and incorrect target labels are chosen at random. This shift represents agents changing their behaviour in an attempt to game the system, becoming bored and clicking answers at random; it is also similar to the situation where a physical agent or sensor moves and can no longer observe the target object.

The pool size is set to 5 agents. For each run, 10 initial responses are drawn for each agent for randomly chosen documents, and the same set of initial responses is supplied to bootstrap all the algorithms tested. These initial responses are theoretically not required to run the Hiring and Firing algorithm or the alternative methods, but saves the computation time of running the algorithms while little information is available to make informed decisions.

1.8.2 Alternative Methods

The Hiring and Firing algorithm (HF) was compared to a simpler method, referred to here as *online screening (OS)*, which is similar to that proposed by [9]. The OS method dynamically tracks the accuracy of agents' responses using DynIBCC, and agents are fired when their accuracy drops below a certain threshold. This can be seen as a simplification of the hiring and firing algorithm, in which the approximate utility is replaced by a scalar accuracy value, independent of the distribution over a task's target decision. For these experiments, the accuracy is calculated using Equation (1.4) applied to the DynIBCC model updated as each response is received. Agents are compared against the unknown agent, whose accuracy is determined from the prior confusion matrices, so is in effect a fixed threshold. If an agent is hired, their next task is chosen at random. Thus the computational cost of OS is lower than that of HF, since it does not calculate the expected utility for different pairs of agents and tasks. Comparing against the online screening approach highlights the advantage of selecting informative tasks for specific agents.

Method Name	Agent Model	Active Selection?	Hiring and Firing?
HF	DynIBCC	Yes	Yes
HFStatic	Static IBCC	Yes	Yes
AS	DynIBCC	Yes	No
OS	DynIBCC	No, random assignment	Yes
Random	DynIBCC	No, random assignment	No

Table 1.5 Features of methods tested for selecting agents and tasks.

We also compared HF with random task selection with no firing (Random), active selection with no firing (AS), and Hiring and Firing using a static agent model (HF-Static). The AS method assigns documents to agents intelligently using the same utility function as Hiring and Firing. However, all original agents are kept on the books and no new agents are recruited. This simpler method does not save any computation costs but is included to show the advantage of replacing agents using HF when they become unreliable. HFStatic uses a static variant of DynIBCC to combine agent responses with text features, which assumes agent reliability is constant [22]. The static method effectively assumes that all responses are made at the same time-step. This allows more rapid computation, since we do not need to account for the dependencies between confusion matrices at each time-step. Hence, the calculations in step 1 of the VB algorithm in Section 1.4.1 can be performed in parallel. Table 1.5 is an overview of the properties of each algorithm. The controlled conditions of the experiment were intended to show the benefits of each property of the complete Hiring and Firing algorithm: the ability to track changing performance; intelligent task selection; and choosing new agents when current agents are not informative.

1.8.3 Results with TREC Documents

Each time new responses were obtained from the simulated crowd, DynIBCC was run to update the combined class decisions (for HFStatic, static IBCC is used instead). The performance was then measured at each iteration by calculating the receiver operating characteristic (ROC) [10] of the combined results, then calculating the area under the curve (AUC). As explained in Section 1.5, the AUC summarises the probability that a randomly chosen positive example is assigned a higher probability than a negative example.

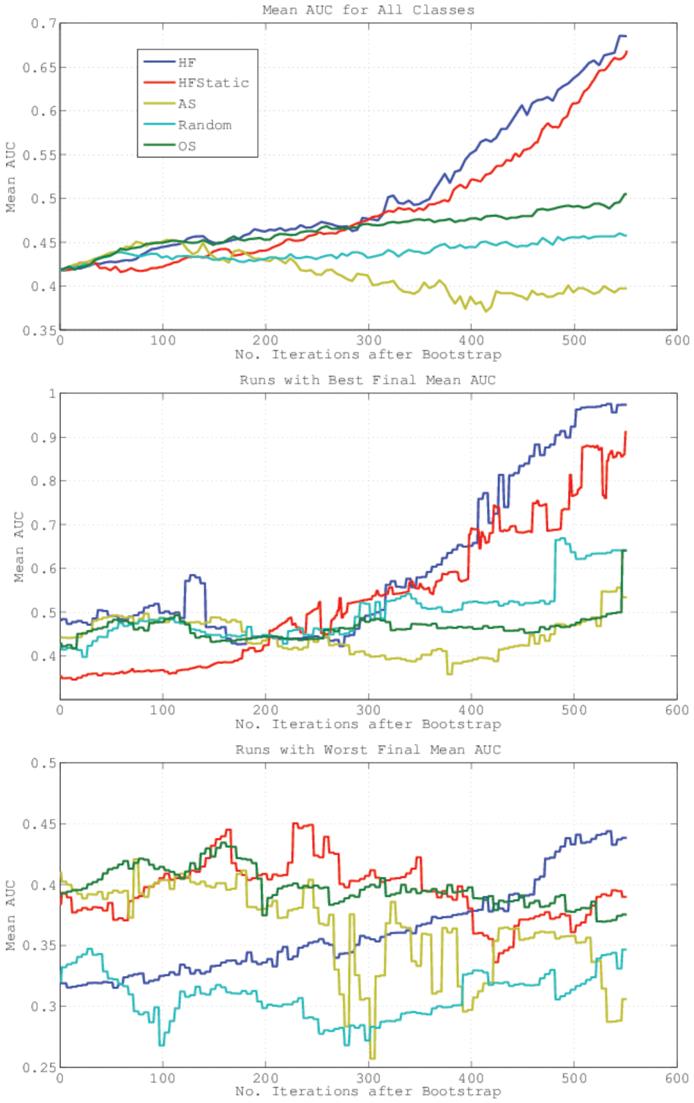

Fig. 1.4 AUC as a function of the number of labels received from agents when using different task assignment algorithms. The top panel shows the mean AUC over 20 runs, the middle panel plots the runs with highest final AUC, and the bottom panel plots the runs with the lowest final AUC. crowdsourced labels are combined with LDA text features from TREC to classify unlabelled documents. Note the faster increase in AUC with Hiring and Firing (HF and HFStatic) compared to active learning with no agent selection (AS), online screening of agents assigned to random tasks (OS), and random assignment with no agent screening (Random). The HF method using a dynamic model of agent behaviour outperforms HFStatic.

Figure 1.4 shows the mean AUC over 20 runs as a function of iterations for the methods in Table 1.5. The HF method has the best performance with a final mean AUC of 0.69, compared to its nearest competitor, the static variant of HF, with 0.67. These two are significantly better than for OS, which does not actively select tasks, with 0.51. Note that for a long period, the mean AUC of all methods is below 0.5 and a lot of time is spent recovering from this position. A difficulty in this experiment is that there were only 37 relevant documents and 600 responses from the simulated crowd, but 2000 LDA features.

After 125 iterations, none of the original set of agents is informative. Examining the mean AUCs in Figure 1.4, the continuing improvement of HF and HFStatic after 125 iterations shows that they must have fired and hired new agents. This contrasts with AS, which does not improve after the agents become uninformative. OS also diverges from HF and HFStatic at 300 iterations, but continues to increase gradually. The Random method diverges from HF and OS around 70 iterations, when some agents start to become uninformative. The AS and Random methods stagnate after a period of time, as they are unable to fire agents and the entire pool eventually becomes uninformative. After 125 labels, all of the original agents are uninformative and AS and Random cannot attain a high AUC. Note that while Random moves closer to 0.5, i.e. expressing complete uncertainty, the AS method decreases to below 0.4 for a period.

In the middle panel of Figure 1.4, we show the best individual run for each method, while the bottom panel shows the worst. This highlights the significant differences in performance between runs. In its best run, HF reaches 0.98 AUC, which follows from a starting AUC close to random around 0.5. In contrast, the worst performance starts with a much lower AUC, near to 0.3, indicating that the bootstrap labels contained a number of errors that result in the model producing the reverse of the correct decisions. The worst-case AUC for HF increases steadily, in contrast to the other methods, which do not show a clear increase in the worst case within 550 iterations. Decreases in the AUC for HFStatic, OS and AS suggest that the responses are consolidating an incorrect model.

The Shannon entropy H can be used to measure the confidence of predictions as new labels are obtained from the crowd, and as such is useful for monitoring an active learning process used by a crowdsourcing system. The total Shannon entropy H_{total} for all document labels $\boldsymbol{t}$, given observations $\boldsymbol{c}$, is defined by

$$H_{total}(\boldsymbol{t}) = -\sum_{i=1}^{N}\sum_{j=1}^{J} p(t_i = j|\boldsymbol{c}) \ln p(t_i = j|\boldsymbol{c}). \tag{1.8}$$

Figure 1.5 shows the mean over 20 runs of the total Shannon entropy of $\boldsymbol{t}$ for the present experiment. All methods see continued decreases in entropy, with HF and HFStatic improving most rapidly. For some runs, the AUCs for AS continued to decrease after the entire agent pool was uninformative; however, the entropy stops decreasing rapidly after 125 iterations, at the point where none of the new crowd responses obtained by AS are informative.

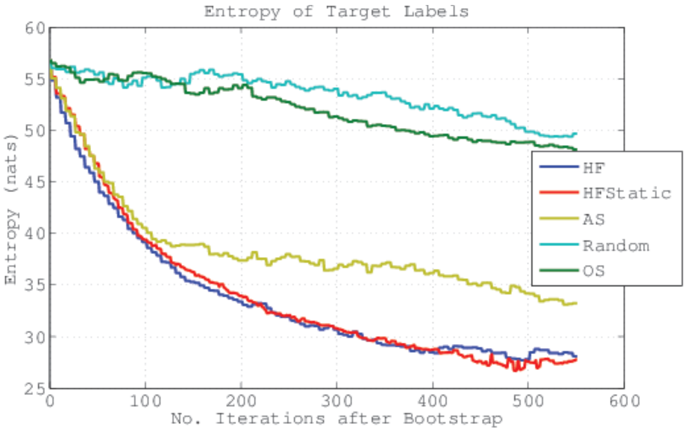

Fig. 1.5 Shannon entropy of the target labels as a function of the number of labels received from the crowd. crowdsourced labels are combined with LDA document features from TREC to classify documents that have not been labelled by the crowd. Entropy is averaged over 20 runs. Note the significantly faster learning rates of the Hiring and Firing approaches (HF and HFStatic) compared to active learning with no agent selection (AS), online screening of agents assigned to random tasks (OS), and random assignment with no agent screening (Random).

1.8.4 Synthetic Dataset

In a separate experiment, the methods HF, HFStatic, OS and Random were re-run over synthetic features to explore whether the LDA features themselves contributed to the variations in performance over multiple runs. It is possible that for some datasets, there were too few features that had sufficient correlation with the target labels. With many unreliable labels and few relevant documents, it is also possible that clusters of negative documents could be identified as the positive group. Synthetic features were drawn from Beta distributions to ensure that the only latent structure in the features related to the target labels. Documents could be relevant to one of three search queries or to none. For each query, there were 15 features with a high probability of values close to one. The remaining 205 features were drawn at random, independent of the query relevance. Hence the features had weak but known correlation with the search queries.

1.8.5 Results with Synthetic Data

Figure 1.6 shows the mean AUCs over 10 runs. Similar patterns are observed for HF and HFStatic as with the LDA features. In the best and worst cases, HFStatic produced better results than HF, although it was worse on average. OS is less extreme

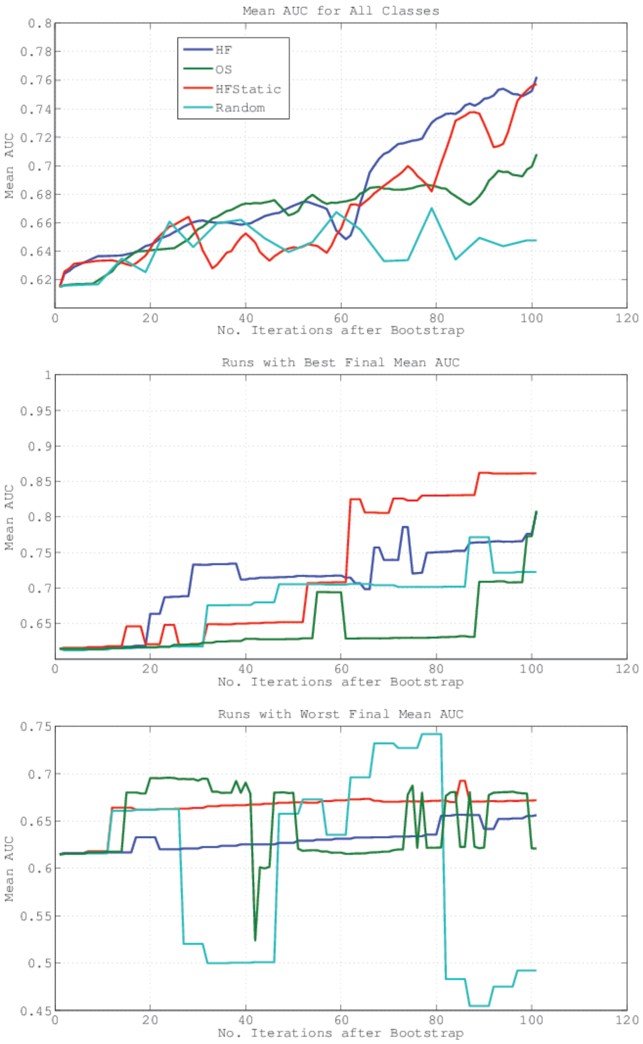

Fig. 1.6 AUC as a function of the number of crowdsourced labels received using different task assignment algorithms. AUC measures classifier performance combining synthetic features with crowdsourced labels. Top panel shows mean AUC over 10 repetitions and 3 topic labels. Middle panel shows the individual run with highest mean AUC over 3 topic labels. Bottom panel shows the run with lowest mean AUC over 3 topic labels. Note the faster increase in AUC with Hiring and Firing (HF and HFStatic) compared to online screening of agents assigned to random tasks (OS), and random assignment with no agent screening (Random). Note also that the HF method, which uses a dynamic model of agent behaviour outperforms HFStatic.

but continues to be overtaken by both Hiring and Firing methods. HF therefore produced the most consistent results.

Figure 1.7 gives an example of the hiring and firing process in HF. The plot shows the approximate utility $\hat{U}(k,i)$ of the optimal task i for three example workers. Agents $k = 1$ and $k = 2$ are reliable throughout the experiment and are hired throughout by all methods. Agent $k = 3$ appears to become gradually less informative until being fired by HF at time step 87. The gradual nature of the change is likely to be because the model requires a number of observations to become certain about the unreliability of the agent, but may also relate to a behaviour change in agent 3.

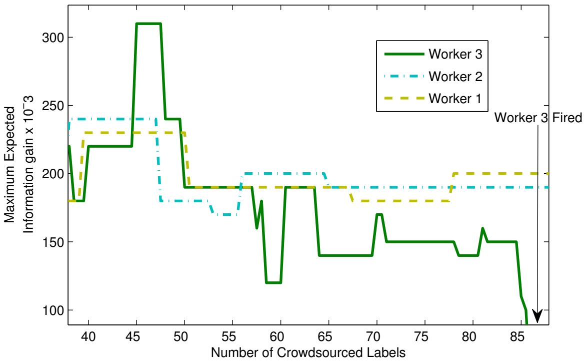

Fig. 1.7 Maximum expected information gain calculated using Equation (1.7) for each of three example agents. The expected information gain is maximised by selecting the most informative task for that particular agent. Note that the expected information gain for agent 3 decreases at first gradually, then more suddenly, before being fired since its expected utility drops below that of hiring a new agent. This decrease occurs either as the system learns more information about the agent, or as their behaviour changes.

1.8.6 Summary of Results

The Hiring and Firing algorithm is the first iteration of the Intelligent Tasking approach, and these simulated experiments demonstrate its advantages over more basic alternatives. HF gains a significant improvement over alternatives through intelligent task selection, hiring new agents effectively, and responding quickly to agent dynamics.

HF and HFStatic have sharper increases in the mean AUCs compared to the OS method, although the latter must replace some unreliable agents since it continues to improve gradually. The different behaviour may result from the hiring and firing algorithms selecting documents intelligently, or from better filtering of uninformative agents. Unlike OS, all three methods using active document selection, i.e. HF, HFStatic and AS, have rapid decreases in entropy, indicating that we obtain a more confident model by selecting tasks intelligently using our expected utility function (see Equation (1.7)). A further advantage of HF over OS may be more clearly realised in a scenario where agents have different skill levels for different types of task. The HF approach uses the full confusion matrix to evaluate agents, rather than the single reliability value used by OS. This enables the system to discriminate between agents with similar overall accuracy but different behaviours and therefore unequal utility.

The variation between best and worst cases suggests that the initial set of responses is critical, particularly in this case where one class occurs much more rarely. It may be possible to improve the hiring and firing algorithm by by selecting the first set of tasks intelligently. To find positive examples more quickly in real-world crowdsourcing systems, we could also introduce weak prior information about the features, for example, by looking at the relationship between features and keywords in the search query. This would allow the intelligent tasking method to select initial sets of documents for crowdsourcing that are more likely to be relevant.

When using the synthetic dataset with 250 features, the differences in performance between each run were less extreme than with 2000 LDA features. This highlights the importance of extracting useful features *a priori*, especially in the absence of training data.

In the average and best cases, HFStatic also improves throughout the experiment, but more slowly than fully dynamic HF. Since the model assumes agents are static, agents that have become uninformative will not be detected until their average confusion matrix over all submitted tasks is worse than that of the unknown agent. In contrast, DynIBCC is able to detect changes rapidly, as shown in [22]. The worst case for HFStatic with TREC data (Figure 1.4) shows the AUC declining over time, which may represent a complete failure to fire uninformative agents. The inclusion of agent dynamics in the DynIBCC model appears to produce more reliable and timely intelligent tasking decisions.

HF reduces entropy at a comparable rate to HFStatic. However, HF uses the DynIBCC model, which has more degrees of freedom than the static IBCC model used by HFStatic, so we might expect it to be more difficult to learn with a high degree of confidence. These results suggests that the more complex DynIBCC model can be learned equally fast as the static model in practical applications.

Further experiments with real agents in a crowdsourcing system are needed to test the performance differences with real behavioural changes and different pool sizes. Ideally, the experiments should be expanded to larger numbers of target values (e.g. more search queries) to better compare the use of confusion matrices with single accuracy measures, such as that used in the OS method.

1.9 Discussion and Future Work

This chapter focused on the efficient deployment of agents for decision-making tasks through the use of descriptive feature data and weak control. First, the potential of DynIBCC to combine continuous-valued features and human agents' responses was shown, demonstrating how this enables efficient analysis of a large dataset in the absence of training examples. Then, an information-theoretic viewpoint was taken to enable the intelligent assignment of tasks to agents in multi-agent systems, resulting in the Hiring and Firing algorithm for crowdsourcing applications such as the TREC challenge. This algorithm was shown to select tasks and agents effectively, outperforming more simplistic approaches. A number of avenues for future work are discussed below, involving the scalability of our algorithms and encouraging more informative agent responses through training and motivation.

Computation time is a significant obstacle that may require more drastic approximations if intelligent tasking is to be applied to larger datasets or to account for future utility. At each iteration, the number of comparisons to be made grows with the number of possible agent-task pairs, but the cost of each comparison also grows with larger problems. First, consider that the number of DynIBCC runs grows linearly with the number of target values (search queries in TREC). Each DynIBCC-VB run consists of a number of iterations, the complexity of which is difficult to describe, partly because it depends on the initial values. With a single new response it is possible to restart the algorithm, adding the new response to the previous data, and since a single update is unlikely to change the variables significantly, we expect to run only a small number of iterations. A single DynIBCC-VB iteration is linear in the total number of agents' decisions plus the number of features multiplied by the number of objects. In the experiments above, documents were clustered to reduce computational cost, which fixes the number of pairs to compare, but does not address the scalability of DynIBCC iterations themselves. This may benefit from further approximating each DynIBCC update.

The document clustering step currently uses a fixed number of clusters, chosen to limit the computation time of each iteration of Hiring and Firing. In future the choice of number of clusters could be treated as a meta-decision, which could be optimised by weighing the expected information gain from using more clusters against the expected time cost and risk of losing agents.

The priors over the confusion matrices provide a benchmark for deciding whether to hire or fire agents, which is fixed before the algorithm is started according to our prior knowledge of similar agents. In future, this prior could be adapted as we observe more agents completing the current set of tasks, which would reduce the need to obtain data to set informative priors when running a new application, and would allow for behavioural shifts in a whole pool of agents. Therefore, a method is required for updating the prior hyperparameters $\boldsymbol{A}_0$ so that the distribution over a new agent's confusion matrix $\boldsymbol{\Pi}^{(k)}$ tends toward the distribution over recently observed agents in the same pool as more such agents are observed.

For reasons of scalability, the algorithm presented here uses a greedy utility function. However, intelligent tasking can naturally be extended to consider rewards over a longer period. This would enable an intelligent tasking algorithm to naturally select training exercises or stimulating tasks that do not have high immediate utility, but may increase the productivity of the agents over the longer term. Future work is needed to investigate methods for inferring the utility of training tasks and the information gained by motivating agents through the choice of task. The information-theoretic framework proposed in this chapter naturally accommodates such enhancements by allowing the improvements in agents to be measured in terms of information learned about the variables of interest. Thus, intelligent tasking forms a principled basis for decision making in information aggregation scenarios.

References

1. Bashir, M., Anderton, J., Wu, J., Ekstrand-Abueg, M., Golbus, P. B., Pavlu, V., and Aslam, J. A. (2012). Northeastern university runs at the TREC12 crowdsourcing track. In *The Twenty-First Text REtrieval Conference (TREC 2012)*. NIST.
2. Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer Series in Statistics. Springer Science+Business Media.
3. Bishop, C. M. (2006). *Pattern recognition and machine learning*. Information Science and Statistics. Springer, 4th edition.
4. Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.
5. Bloodgood, M. and Callison-Burch, C. (2010). Using Mechanical Turk to build machine translation evaluation sets. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazons Mechanical Turk*, pages 208–211. Association for Computational Linguistics.
6. Chen, X., Bennett, P. N., Collins-Thompson, K., and Horvitz, E. (2013). Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 193–202. ACM.
7. Dasgupta, S. (2004). Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17*, pages 337–344. MIT Press.
8. Dawid, A. P. and Skene, A. M. (1979). Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):20–28.
9. Donmez, P., Carbonell, J., and Schneider, J. (2010). A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *SIAM International Conference on Data Mining (SDM)*, pages 826–837. Society for Industrial and Applied Mathematics.
10. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874.
11. Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721 741.
12. Harris, C. and Srinivasan, P. (2012). Using hybrid methods for relevance assessment in TREC Crowd '12. In *The Twenty-First Text REtrieval Conference (TREC 2012)*. NIST.
13. Hu, Q., Xu, Z., Huang, X., and Ye, Z. (2012). York university at TREC 2012: Crowdsourcing track. In *The Twenty-First Text REtrieval Conference (TREC 2012)*. NIST.
14. Ipeirotis, P. G., Provost, F., and Wang, J. (2010). Quality management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 64–67. ACM.

15. Kamar, E., Hacker, S., and Horvitz, E. (2012). Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '12*, pages 467–474. International Foundation for Autonomous Agents and Multi-Agent Systems.
16. Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.
17. Levenberg, A., Pulman, S., Moilanen, K., Simpson, E., and Roberts, S. (2014). Economic Indicators from Web Text Using Sentiment Composition. *International Journal of Computer and Communication Engineering.*
18. Liu, Q., Peng, J., and Ihler, A. (2012). Variational inference for crowdsourcing. In *Advances in Neural Information Processing Systems 25*, pages 701–709. MIT Press.
19. Nellapati, R., Peerreddy, S., and Singhal, P. (2012). Skierarchy: Extending the power of crowdsourcing using a hierarchy of domain experts, crowd and machine learning. In *The Twenty-First Text REtrieval Conference (TREC 2012).* NIST.
20. Quinn, A. J., Bederson, B. B., Yeh, T., and Lin, J. (2010). Crowdflow: Integrating machine learning with Mechanical Turk for speed-cost-quality flexibility. *Technical Report HCIL-2010-09, University of Maryland, College Park.*
21. Raykar, V. C. and Yu, S. (2012). Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13:491–518.
22. Simpson, E. and Roberts, S. and Psorakis, I. and Smith, A. (2013). Dynamic Bayesian mombination of multiple imperfect classifiers. In *Decision Making and Imperfection*, pages 1–35. Springer.
23. Simpson, E., Reece, S., Penta, A., Ramchurn, G., and Roberts, S. (2013). Using a Bayesian model to combine LDA features with crowdsourced responses. In *The Twenty-First Text REtrieval Conference (TREC 2012), Crowdsourcing Track.* NIST.
24. Smith, A. M., Lynn, S., Sullivan, M., Lintott, C. J., Nugent, P. E., Botyanszki, J., Kasliwal, M., Quimby, R., Bamford, S. P., Fortson, L. F., Schawinski, K., Hook, I., Blake, S., Podsadlowski, P., onsson, J. J., Gal-Yam, A., Arcavi, I., Howell, D. A., Bloom, J. S., Jacobsen, J., Kulkarni, S. R., Law, N. M., Ofek, E. O., and Walters, R. (2010). Galaxy Zoo Supernovae. *Monthly Notices of the Royal Astronomical Society.*
25. Smith, A. and Lintott, C. (2010). Web-scale citizen science: from Galaxy Zoo to the Zooniverse. In *Proceedings of the Royal Society Discussion Meeting 'Web Science: A New Frontier'.* The Royal Society.
26. Smucker, M. D., Kazai, G., and Lease, M. (2012). Overview of the TREC 2012 crowdsourcing track. In *The Twenty-First Text REtrieval Conference (TREC 2012).* NIST.
27. Smucker, M. D., Kazai, G., and Lease, M. (2012). TREC 2012 crowdsourcing track TRAT task results. In *The Twenty-First Text REtrieval Conference (TREC 2012).* NIST.
28. Yan, Y., Fung, G. M., Rosales, R., and Dy, J. G. (2011). Active learning from crowds. In *Proceedings of the 28th International Conference on Machine Learning*, ICML '11, pages 1161–1168.
29. Zhang, C., Zeng, M., Sang, X., Zhang, K., and Kang, H. (2012). BUPT_PRIS at TREC 2012 crowdsourcing track 1. In *The Twenty-First Text REtrieval Conference (TREC 2012).* NIST.