Active Ordering of Interactive Prediction Tasks

Abhimanyu Lad
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
alad@cs.cmu.edu

Yiming Yang
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
yiming@cs.cmu.edu

Abstract

Many applications involve a set of prediction tasks that must be accomplished sequentially through user interaction. If the tasks are interdependent, the order in which they are posed may have a significant impact on the effective utilization of user feedback by the prediction systems, affecting their overall performance. This paper presents a novel approach for dynamically ordering a series of prediction tasks by taking into account the effect of user feedback on the performance of multiple prediction systems. The proposed approach represents a general strategy for learning incrementally during test phase when the system interacts with the end-user, who expects good performance instead of merely providing correct labels to the system. Therefore, the system must balance system benefit against user benefit when selecting items for user's attention. We apply the proposed approach to two practical applications that involve interactive trouble report generation and document annotation, respectively. Our experiments show significant improvements in prediction performance (in terms of Mean Average Precision) using the proposed active ordering approach, as compared to baseline approaches that either determine a task order offline and hold it fixed during test phase, or do not optimize the order at all.

1 Introduction

Much research in statistical learning has focused on the optimization of classification, regression, recommendation, and retrieval systems in isolation. However, in many practical applications, the user needs to interact with multiple prediction systems to accomplish a higher level goal, e.g., interacting with multiple decision support systems to resolve each trouble report in a troubleshooting system, or annotating each text document with respect to multiple classification taxonomies, and so on. For each input instance, the user must accomplish a series of tasks, assisted by a set of prediction systems that provide useful suggestions for each task and receive corresponding feedback from the user. If the tasks are interrelated, as is often the case in such scenarios, the user feedback received for each task at *run-time* (or test phase) can potentially help in improving performance on subsequent prediction tasks. Therefore, the order in which the tasks are posed may have a significant impact on the effective propagation of user feedback through the prediction tasks, affecting their overall performance. How to find such an order for multiple prediction tasks to maximize their overall performance is an open challenge that has not received attention in the research community.

As an illustrative example, consider the problem of assigning multiple interrelated categories or labels to documents, which correspond to multiple interrelated prediction tasks. In a batch setting, the system would make all category assignments to each document without any user intervention, allowing the user to check and modify the assignments only after all predictions have been made. However, the interrelated nature of the prediction tasks suggests a more efficient approach for minimizing user effort: The system proceeds in a stepwise manner by predicting one label at a time, which the user can immediately validate and modify if required. This user feedback provides additional information that can be used by the system before it predicts the next label for the same document. Depending on the nature of dependencies among categories, the system has the opportunity to choose the order of these predictions so that the overall prediction accuracy is maximized and the total user annotation effort is minimized. The effect of prediction order on overall performance was also observed by Read et al. [19], who proposed a "classifier chaining" approach for multi-label classification. However, they did not solve the problem of finding the best chain (i.e., the order of predicting class labels), and instead, circumvented the problem by using an ensemble of randomly chosen permutations.

Another example that involves heterogeneous (classification, regression, and retrieval) predictions is interactive trouble report management. Each trouble report from the customer triggers a sequence of interdependent decisions, e.g., determining the priority, category, subcategories, hardware type, and software type, assigning an expert, and finally formulating a resolution. For making each of these decisions, the user would be assisted by the system, which makes helpful predictions for each task and receives corresponding user feedback.

It might be possible to come up with an ordering of these decisions that improves the overall accuracy of the predictions through more efficient use of the user's feedback. For example, it might be beneficial to finalize the hardware and software type of the problem before assigning an expert, because historical data suggests that the latter decision depends heavily on the former. By doing so, the system would be able to make more accurate predictions for the expert assignment. How to optimize the task order by simultaneously taking into account all such relationships and dependencies is an open challenge.

To the best of our knowledge, the problem of how to order a set of interrelated prediction tasks to optimize their overall performance has not been studied in the literature, except for our previous work in a recent report [12], where we formulated the problem as that of *static task ordering*, i.e., using past data in an *offline* phase to find a task order that has the best performance on average, and then holding this order fixed for all data instances at run-time. This approach has the advantage that a good task order is induced from a large corpus of past data, thus allowing effective discovery of task order preferences that are inherent in the domain at hand. However, the search for the best order requires the solution to an NP-hard problem, which makes this approach unscalable to domains with numerous prediction tasks. Moreover, the use of a fixed order makes this approach insensitive to the peculiarities of individual input instances where a different task order would be more appropriate.

The focus of this paper is to go beyond offline approaches, which find task orders that are optimized for the average case, and instead, dynamically choose the task order for good performance on each input instance. Such an approach must be able to evaluate the effect of various task orders on the predictive performance, within the context of the current input instance. Our approach for estimating the benefit of user feedback with respect to different task orders is inspired by uncertainty sampling based active learning methods that use the learning algorithm's *confidence* on data points as the criterion for choosing the next query for the user [7, 15]. However, our problem setup is significantly different in important ways. Traditional active learning optimizes the order in which instances are labeled to improve the learning curve of a single prediction system during *training phase*. However, our focus is on the *test phase*, i.e., when the system interacts with the end-user, who is not concerned with training the system, but is only interested in accomplishing a series of tasks with minimal effort. Therefore, the system is expected to make useful predictions, whose accuracy is assessed before the correct label is made available in the form of user feedback. Hence, the task ordering strategy must strike a balance between the current predictability and the future utility of candidate queries, since the most informative labels are usually the ones that the current model is most uncertain about. Unfortunately, choosing queries that are expected to provide maximum system benefit are least likely to provide immediate user benefit. This observation leads to another view of our problem setup − *active learning during test phase*. Consider a collaborative filtering system that uses active learning strategies to improve its model of the user's preferences. When the system is in the training phase (e.g., *interview phase* for a new user), it is reasonable to probe the user with items that the system is most uncertain about. However, such a separate interview phase might not be practical in a deployed system that is perpetually in the test phase, when users expect good performance from the beginning of their interaction with the system. This leads to an exploration-exploitation trade-off similar to that in reinforcement learning: The goal of maximizing overall user benefit requires the system to strike a balance, at each step, between (i) immediate user benefit obtained by exploiting the current model and making the best available prediction, and (ii) system benefit obtained by probing the user with queries whose outcomes are uncertain, in a hope to improve future user benefit. However, such a setup would still focus on a single prediction model. In this paper, we consider multiple interrelated prediction models, where the user feedback received on one model's output can potentially benefit other prediction models.

Our step-wise prediction process is similar to a model of inductive learning known as *online learning* [17], where the training phase is not separate from the prediction phase. Instead, the system learns one instance at a time: It makes a prediction on a single instance, receives feedback, which it uses to update its hypothesis before moving onto the next instance. However, *online learning* focuses on a single prediction problem, while we focus on multiple interrelated prediction problems; The feedback received on the outcome of one prediction task is used to improve the performance of the other prediction tasks. Moreover, *online learning* does not deal with the problem of proactively choosing the order of predictions.

Multi-task learning [2, 29] deals with simultaneously learning multiple prediction tasks by leveraging their structural similarities in the functional space. However, the focus is clearly on the *training phase*. During *test phase* or prediction phase, the tasks are assumed to be performed in isolation, independently from each other and without any user interaction, which makes the order of prediction irrelevant, and therefore, fails to leverage tasks dependencies to improve performance at run-time. Similarly, *transfer learning* [23] focuses on transferring knowledge gained during learning one prediction task to help in learning another task in the same or related domain. However, the transfer occurs in the *training phase*, ignoring the information that can be transferred at run-time, when the end-user interacts with the system. Moreover, the direction and sequence of the transfer is not considered, assuming such a choice is indeed available for a given set of learning tasks.

We seek to address the task ordering problem in the context of a multi-step interactive process with the following properties: (i) the multiple predictions are interdependent and the feedback (user correction) received for a prediction task can be incorporated by subsequent tasks at run-time, (ii) the order of the prediction tasks is flexible, free from user interface constraints, and (iii) the optimal order of the tasks is not obvious to the user due to a large number of subtle dependencies that exist among the prediction tasks.

In the next section, we formalize the task ordering problem and distinguish between static and active task ordering strategies. In Section 3, we develop a active task ordering strategy and show how it can be combined with static ordering. We describe our experimental setup including datasets and evaluation metrics in Section 4. In Section 5, we describe the results of our comparative evaluation of various task ordering strategies, and also the results of additional experiments performed to analyze the effectiveness of the proposed approach. We discuss the assumptions and limitations of the proposed approach in Section 6 and conclude the paper in Section 8.

2 The Task Ordering Problem

Consider a set of N input instances or "documents", and for each document d_i, we have K prediction tasks $\{t_1^{(i)}, t_2^{(i)}, ..., t_K^{(i)}\}$. Such data can be represented using a $N \times K$ matrix, where rows denote documents and columns denote the outcomes or target values of the prediction tasks, e.g., multiple categories, numerical or ordinal values for each document. This is a natural representation for many multi-prediction setups, for instance, the problem of assigning multiple categories to each document (e.g., *polycategorization* [26]). Another example is collaborative filtering, where rows denote users and columns denote items, and each cell in the matrix denotes a user's rating of an item. In these examples, the goal is to fill the empty cells based on observed cells in a partially-filled matrix or a separate completely-filled matrix of training data.

In this paper, we consider an *interactive* or *online* version of the above-mentioned prediction problems: For a given row, the system makes prediction for a single column at each step. The user validates or corrects the prediction, and the system takes this user feedback into account before moving to the next column. We allow the system to re-order the columns for each row independently, i.e., the system decides the order of the predictions, and hence, the order in which user feedback is received.

We are now ready to formalize the task ordering problem. Let π denote a bijection of $\{1, 2, ..., K\}$ onto itself, such that $t_{\pi(k)}^{(i)}$ denotes the k^{th} task in the permutation induced by π for the i^{th} document. In the k^{th} step of the prediction process, the system performs prediction task $t_{\pi(k)}^{(i)}$, based on the (system-predicted or user-corrected) outcomes of the tasks already visited by the system in the previous steps, $\{t_{\pi(1)}^{(i)}, t_{\pi(2)}^{(i)}, ..., t_{\pi(k-1)}^{(i)}\}$.

Let the prediction performance in the k^{th} step be denoted by $\Delta(\hat{t}_{\pi(k)}^{(i)}, t_{\pi(k)}^{(i)})$, where the function $\Delta(\hat{y}, y)$ scores the prediction $\hat{y}$ with respect to the truth y. It can be one of the common measures like classification error or squared error, or some other metric that is appropriate for the given domain.

The task ordering problem can then be formulated as follows: For each document d_i, find the permutation π_i^* that maximizes the overall performance on all prediction tasks (columns) for d_i:

$$\pi_i^* = \arg\max_{\pi} \sum_{k=1}^{K} \Delta(\hat{t}_{\pi(k)}^{(i)}, t_{\pi(k)}^{(i)}) \tag{2.1}$$

where $t_{\pi(k)}^{(i)}$ denotes the k^{th} prediction task for the i^{th} document according to the permutation π.

The number of possible permutations is factorial in the number of tasks, and the overall performance of any given permutation depends heavily on the behavior of the various prediction systems as well as the performance metric employed. Hence, the true value of the objective function can only be calculated empirically rather than analytically. Therefore, evaluating or searching through all permutations is computationally prohibitive, leading to a hard optimization problem. In order to proceed, we must explore methods that approximate or indirectly optimize the objective function. We divide the approaches into two broad categories: *static ordering* and *active ordering* approaches.

2.1 Static Task Ordering In our previous work [12], we explored a solution that circumvents the computational issue in two ways: (i) instead of finding the best task order for each new input instance, we used the behavior of the prediction systems on past data to derive a single order in an offline phase that has the best performance *on average*, and then held it fixed for all data instances at run-time, and (ii) we approximated the objective function in terms of pairwise dependencies among tasks. However, combining the pairwise preferences to deduce an optimal total ordering of the tasks still required the solution to the well-known Linear Ordering Problem (LOP) [20], which is known to be NP-hard [5]. (See [12] for technical details of the approach).

The *static task ordering* approach optimizes an approximation of the following objective function:

$$\pi^* = \arg\max_{\pi} \ \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\Delta(\hat{t}^{(i)}_{\pi(k)}, t^{(i)}_{\pi(k)})$$

Thus, it learns a single task permutation π^*, instead of one for each input instance. In other words, all π^*_i correspond to the same permutation, denoted by π^*. This approach is not sensitive to the current data instance and hence π^* is not guaranteed to perform well for each instance at run-time.

2.2 Active Task Ordering The focus of this paper is to devise strategies for finding task permutations that are instance specific, i.e., task orders that are optimized for performance on each given document. The task ordering is induced greedily: At each step, the system only has to choose the next prediction task, based on outcomes of the tasks already solved. We call this approach *active task ordering.*

Algorithm 1 shows how user interaction proceeds. We start with trained classifiers[1] $\mathcal{C}_1, ..., \mathcal{C}_k, ..., \mathcal{C}_K$ for each of the K tasks, and the ordering strategy Φ. Each such classifier has an input feature space comprising the outcomes of the rest of the tasks, with missing values for tasks that have not been visited yet. (See Section 4.3 for classifier implementation details.)

Let symbols with upper case T denote task (or column) identities, e.g., T_k denotes the k^{th} task, and symbols with lower case t denote the corresponding outcome of the task. At each step, the ordering strategy chooses the next task to be accomplished. Then the corresponding classifier generates a prediction for that task. If this prediction is correct, no action is required on the part of the user. Otherwise, the user provides the correct value; the system updates the value for that task accordingly and also receives a penalty for the incorrect response.

Algorithm 1 Steps involved in the interactive process with active task ordering. T_k denotes the k^{th} prediction task; t_k denotes the outcome of the k^{th} task.

```
Require: Trained classifiers $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k$
Require: Ordering strategy $\Phi$
 1: for each document do
 2:     for $k$ in 1 to $K$ do
 3:         $T_k \leftarrow \Phi(t_{1...(k-1)})$
 4:         $\hat{t_k} \leftarrow \mathcal{C}_k(t_{1...(k-1)})$
 5:         Make prediction $\hat{t_k}$ for task $T_k$
 6:         Solicit correct value $t^*_k$ from user
 7:         Update performance $\Delta(\hat{t_k}, t^*_k)$
 8:         $t_k \leftarrow t^*_k$ { Assign correct value }
 9:     end for
10: end for
```

3 Proposed Approach

We desire an ordering strategy Φ that, at step k, takes into account the outcomes of tasks already solved, and outputs the next task that should be brought to the user's attention:

$$\Phi(t_{1...(t-1)}) \ \rightarrow \ T_k$$

As shown in Algorithm 1, when the system chooses a task for the user's attention (Step 4), it must also be able to make a prediction for that task (Step 5), since the system's performance will be evaluated on this prediction (Step 7) before the correct value is added to the feature set (Step 8) of the subsequent prediction tasks.

Therefore, the ordering strategy must choose the next task such that the corresponding classifier can make an accurate prediction for that task (thus, benefitting the user), and whose correct label, once verified/entered by the user, would be useful for subsequent predictions (thus, benefitting the system). In other words, the ordering strategy must choose the next task at each step based on two criteria:

1. **The predictability of the task**, i.e., how well the system, in its current state, can perform on the chosen task.

2. **The future utility of the task**, i.e., how helpful the knowledge of the task's outcome will be for improving the predictions on subsequent tasks.

Let the predictability and utility of task T_k at step k be denoted by $\mathcal{P}(T_k)$ and $\mathcal{U}(T_k)$, respectively. Then, we define our active task ordering strategy as choosing the next task to be the one that maximizes a linear

[1] We use the term *classifier* to refer to any kind of prediction system, including classification, regression or retrieval systems.

combination of predictability and utility:

$$\Phi(t_{1\ldots(k-1)}) = \arg\max_{T_k} \{\lambda\mathcal{P}(T_k) + (1-\lambda)\mathcal{U}(T_k)\} \tag{3.2}$$

where λ controls the relative influence of the two criteria.

Note that both criteria depend on assessing how the system will perform on a prediction task. However, this is impossible to measure before the true label is received from the user, which can only happen after the system chooses a task for the user's attention. In order to break this deadlock, a common approach in active learning is to use the *confidence* of the prediction system as a surrogate for the expected performance of the classifier on a data point. In the context of classifiers, various definitions of confidence exist. For probabilistic classifiers, probability values away from 0.5 denote more confident predictions [14]. "Entropy" extends the same idea to multi-class prediction problems. For non-probabilistic classifiers like SVMs, distance of data point from decision boundary has been known to be an effective measure of confidence (or conversely, *uncertainty*) with desirable properties in the context of active learning [25]. For now, we avoid the exact definition of *confidence* (see Section 4.4 for details), and denote it by $\kappa(\mathcal{C}, X)$, i.e., the confidence of the classifier $\mathcal{C}$ with respect to the data point X.

The predictability $\mathcal{P}(T_k)$ is straightforward to define in terms of the confidence of task T_k's classifier:

$$\mathcal{P}(T_k) = \kappa(\mathcal{C}_k, \{t_{1\ldots(k-1)}\})$$

since the input data available to the classifier at step k is simply the outcomes of the tasks already solved till step $k-1$.

The utility $\mathcal{U}(T_k)$ is defined as the increase in confidence on the remaining tasks if the true label of task T_k, denoted by t_k, were known and added as input to the rest of the prediction tasks:

$$\mathcal{U}(T_k) = \sum_{j\in\mathcal{T}'} \kappa(\mathcal{C}_j, \{t_{1\ldots k}\}) - \kappa(\mathcal{C}_j, \{t_{1\ldots(k-1)}\})$$

where $\mathcal{T}'$ denotes the indices of tasks that would still remain to be accomplished, if task T_k were to be selected at step k. That is, $\mathcal{T}' = \{T_{1\ldots k}\} \setminus \{T_{1..(k-1)}, T_k\}$.

However, the true outcome t_k is unknown before we actually make the decision to select T_k as the next prediction task. For probabilistic classifiers, a common approach in active learning is to sum over all possible labels weighted by their respective probabilities, as given by the current classifier, to obtain an *expectation* over the quantity of interest. However, this might not be practical for classification tasks admitting multiple labels simultaneously, leading to a sum over an exponential number ($2^{|T|}$) of possible labelings. In such a case, one can simply use the most probable labeling as generated by the current classifier. We make this choice due to the presence of highly multi-valued prediction tasks in our datasets (Section 4.1).

3.1 Combining Active and Static Ordering

The static ordering approach (Section 2.1) has the advantage of being induced from a large corpus of past data, whereas the active approach only uses the current document to deduce the order and therefore might suffer from data sparsity problems. Moreover, the static approach has the ability to directly optimize for the evaluation metric [12], whereas the active approach is dependent on a notion of classifier confidence that is only indirectly related to the evaluation metric. On the other hand, the active approach is sensitive to the current input instance (i.e., the outcomes of the tasks already visited) whereas the static approach uses the same task order for all documents at run-time.

Therefore, a promising middle ground would be to combine the two approaches; specifically, use the static ordering as a *prior* in the active ordering strategy. The static order would serve to reduce the variance of the active strategy in light of sparse data, and also allow the overall ordering to be sensitive to the target evaluation metric.

Let π denote the static order induced using the methods described in [12]. We modify equation 3.2 as follows:

$$\Phi(t_{1\ldots(k-1)}) = \arg\max_{T_k} \mathcal{S}(T_k)\cdot\{\lambda\mathcal{P}(T_k) + (1-\lambda)\mathcal{U}(T_k)\} \tag{3.3}$$

where the static order prior $\mathcal{S}(T_k)$ is defined as:

$$\mathcal{S}(T_k) = \exp(-\sigma\cdot\pi^{-1}(T_k))$$

$\pi^{-1}(T_k)$ is the position of task T_k if the remaining tasks (i.e., tasks not yet visited by the active order) were ordered according to the static permutation π. Thus, at any given point in the active order, the static order prior favors candidate tasks that occurred earlier in the static order, but have not been included in the active order yet. σ is a tunable parameter that controls the strength of the prior.

4 Experimental Setup

Our experimental goals are two-fold: (i) To compare the performance of various task ordering approaches against each other, and also against unoptimized task orders, and (ii) to study the behavior of task ordering in general and understand how and why it affects the overall performance of the prediction systems.

Table 1: The Accenture Dataset

Attribute Name	Distinct Values
Abstract	*Free text*
Title	*Free text*
TopicTags	81
BusinessFunctionKeywords	21
PertinentToOrgUnit	20
Keywords	20
IndustryKeywords	20
ItemType	17
Offerings	17
TechnologyKeywords	14
Client	11
PertinentToServiceLine	11
VendorProductKeywords	9

Table 2: The F/A-18 Dataset

Attribute Name	Distinct Values
Problem Summary	*Free text*
PointOfContact	34
Base	31
Application	24
ScheduleTo	20
Category$_1$	17
Priority	8
HardwareType	7
Category$_2$	7
AssetStatus	6
OperatingSystem	4
Source	4
ReportStatus	4

To this end, we use two datasets, both in the form of documents with multiple attributes. Predicting each of these attributes based on past information as well as user feedback corresponds to the multiple prediction tasks. The goal is to re-order these tasks so as to maximize the overall accuracy of the predictions produced by the system. In this paper, we restrict our attention to classification tasks only (see Section 6 for further discussion).

4.1 Datasets The first dataset was collected from Accenture, a large consulting corporation. It consists of 60,000 documents related to client projects. Each document has associated meta-data in the form of 13 attributes, and predicting the values of these attributes corresponds to the prediction tasks. *Abstract* and *Title* are treated as the initial text input. Table 1 describes the structure of this dataset.

The second dataset was collected from Inmedius, which provides logistics support and publication tools to technicians who maintain U.S. Navy's F/A-18 aircraft. The data consists of 6,000 aircraft trouble reports, and each report contains 13 attributes that correspond to the various decisions made by operators in the process of resolving a trouble report. *Problem summary* is treated as the initial text input, and the rest of the 12 attributes are predicted in a stepwise manner. Table 2 describes the structure of this dataset.

We divide both datasets into three equal parts:

1. **Classifier training set**: This portion was used to train and tune the classifiers.
2. **Order training set**: This portion was used to estimate a *static* task order [12]. Note that this subset is not required by the active ordering approach described in this paper, since there is no learning phase as far as the ordering strategy is concerned. Instead, we use this subset to tune parameters like λ and σ (Section 3) in the active ordering approach. The exact values of the tuned parameters depend on the dataset as well as the target performance metric, but the optimal values were found to be in the following ranges: λ: 0.5 to 0.75, and σ: 0.1 to 0.2.
3. **Test set**: This portion was used to evaluate the performance of the different task ordering strategies.

We switch the roles of the three parts and calculate the average scores on the 6 combinations, to get a more reliable picture of the algorithms' performance.

4.2 Evaluation Metrics The goal of the system is to minimize human effort by producing accurate predictions for all the tasks, which would directly translate to less user intervention for replacing incorrect responses. We use two evaluation metrics that capture slightly different aspects of the accuracy of the system's predictions: Mean Average Precision (MAP), and F_1.

MAP [1] is a rank-based metric that measures the quality of the ranked list produced by a system. It is the average precision at all recall points in the ranked list. We use MAP to measure the ability of the system to present its predictions in the form of a ranked list such that the correct answers appear near the top of the list for easy selection by the user.

F_1 is a set-based metric, equal to the harmonic mean of Recall (fraction of relevant items retrieved) and Precision (fraction of retrieved items that are relevant) [27]. We report both macro-averaged (averaged over categories) as well as micro-averaged (averaged over total documents) F_1 scores. The macro-average is dominated by performance on rare labels, while micro-average is dominated by performance on labels that appear in many data instances [13]. We use F_1 to measure the ability of the system to present its predictions in the form of binary (yes/no) decisions (e.g, whether a document belongs to a particular category, as opposed to simply ranking all the categories).

4.3 Classifiers We choose Linear Support Vector Machines (SVMs) for all our experiments because of their superior performance on many classification problems [3] and the ability to handle a large number of potentially redundant features, which is common in the domain of text classification [11]. One SVM classifier is trained per attribute to be predicted, treating it as the target variable, while the rest of the attributes are treated as the predictors. At each step in the prediction process, only a subset of the predictors would be available, which correspond to the tasks that have already been visited, and hence, whose outcomes are known. The rest of the predictors would have *missing values*, which we impute using the nearest-neighbor approach [6]. A score-based thresholding strategy with a five-fold cross-validation was used to convert the scores produced by the classifiers into decisions [28]. Since each attribute can take multiple values, the prediction task is a multi-label classification problem, which we address using a one-vs-all scheme [21].

4.4 Estimating Prediction Confidence For the definition of classifier *confidence*, as required in Section 3, we use distance of the data point from SVM's decision boundary. In the context of active learning, choosing the next *query* as the data point closest to the decision boundary has been shown to be an effective approximation for the strategy of choosing successive queries that reduce the size of the version space of SVM classifiers the most [25]. In the standard active learning setup for SVM, there would be a single classifier and multiple candidate data points. Hence, calibration of the classifier's scores would be unnecessary since the calibrated scores would still be proportional to the original outputs of the classifier for each data point. However, in our case, the confidence scores for different tasks, i.e., different SVM classifiers in our case, must be comparable to each other. Therefore, we convert all SVM scores to probability estimates using LIBSVM [4], which employs an improved implementation [16] of Platt's calibration algorithm [18].

Since we are using a one-vs-all scheme for multi-valued predictions, an attribute that takes V values will require V corresponding binary classifiers. We must combine the confidence scores of each of these classifiers to derive a single confidence score for the prediction task. How to do this for classification problems with multiple overlapping labels is not entirely clear, therefore we resort to heuristics. Let the calibrated scores of the V classifiers for an attribute be denoted by p_v where $v = 1, ..., V$. We examined the following combination methods for computing confidence:

`Mean`: average deviation from 0.5:

$$\kappa = \frac{1}{V}\sum_{v=1}^{V} |p_v - 0.5|$$

`GeoMean`: geometric mean of the deviations from 0.5:

$$\kappa = (\prod_{v=1}^{V} |p_v - 0.5|)^{\frac{1}{V}}$$

`Min`: minimum deviation from 0.5:

$$\kappa = \min_{v=1..V} |p_v - 0.5|$$

`Max`: maximum deviation from 0.5:

$$\kappa = \max_{v=1..V} |p_v - 0.5|$$

`Variance`: variance of the scores:

$$\kappa = \frac{1}{V}\sum_{v=1}^{V} (p_v - \bar{p_v})^2$$

`Mean` and `GeoMean` take into account all the confidence scores of the classifiers for an attribute, whereas `Min` and `Max` look at the extreme (lowest or highest) confidence scores of the classifiers. `Variance` looks at the spread of the scores, which might be a good indicator of how confident the set of classifiers are in ranking the labels: Larger spreads signify more confident rankings, while smaller spreads signify uncertainty in ordering the labels. (See Section 5.2 for an empirical comparison).

5 Results

We conducted a comparative evaluation of various ordering approaches and report the results in Section 5.1. We also conducted additional experiments to compare the various alternatives for estimating classifier confidence (Section 5.2), and also to understand the effect of task order on predictive performance (Section 5.3).

Table 3: Performance of various task ordering approaches on the Accenture dataset

	Evaluation Metric		
Approach	MicroF1	MacroF1	MAP
COMBINED	**0.6447**	0.4326	**0.7967**
ACTIVE	0.6402	0.4311	0.7926
STATIC	0.6355	**0.4391**	0.7691
Random	0.5343	0.3066	0.7194

Table 4: Performance of various task ordering approaches on the F/A-18 dataset

	Evaluation Metric		
Approach	MicroF1	MacroF1	MAP
COMBINED	**0.8625**	0.5714	**0.9289**
ACTIVE	0.8611	**0.5715**	0.9265
STATIC	0.8550	0.5701	0.9032
Random	0.7983	0.5352	0.8990

5.1 Comparison of Ordering Approaches Table 3 and 4 show the results obtained using various task ordering approaches on the Accenture and F/A-18 datasets, respectively. The task ordering approaches include: STATIC: static task ordering [12], ACTIVE: active task ordering *without* using the static order as prior (i.e., equation 3.2), and COMBINED: active task ordering with the static order as prior (i.e., equation 3.3). As a baseline approach, we use Random task orderings, i.e., arbitrary task permutations for each document.

On both datasets, the task ordering approaches beat the baseline approach by a significant margin with respect to all three performance metrics. The active ordering approaches proposed in this paper (ACTIVE and COMBINED) beat our previous approach (STATIC) by a significant margin in terms of MAP. However, the performance gain in terms of MicroF1 is relatively small, while the differences in terms of MacroF1 are mostly insignificant. Although the improvement of ACTIVE over STATIC is not drastic, it should be noted that the STATIC ordering approach requires a large training corpus as well as the solution to an NP-hard problem, whereas the ACTIVE ordering approach is an online algorithm that only looks at the current input instance, and provides a much more efficient and scalable solution for task ordering when the number of tasks is large, or training data (as required by the STATIC approach) is at a premium.

Significance Tests: To evaluate the differences between classifier performances with respect to different task orderings, we conducted a *sign test* for each pair of methods listed in Table 3, and similarly for Table 4, with respect to the MAP scores of all predictions (i.e., number of documents × number of attributes per document). On the Accenture dataset, differences in MAP scores greater than 0.005 were found to be highly statistically significant (p-value $\ll 0.01$). On the F/A-18 dataset, differences in scores greater than 0.003 were found to be highly statistically significant (p-value $\ll 0.01$).

5.2 Correlation between Prediction Confidence and Actual Performance In our active ordering approach, we have used *prediction confidence* as a surrogate for estimating the system's performance on a prediction task (see Section 3). We would like to assess how well our estimated prediction confidence correlates with the true performance, measured when the correct label is obtained from the user. We use a held-out subset of the Accenture dataset and make the system generate a confidence score in addition to its prediction, and then measure the correlation of its confidence score against the performance score.

Table 5 shows the correlation between various formulations of prediction confidence (as described in Section 4.4) and two performance metrics: MAP and Accuracy[2]. When using MAP, which is a rank-based evaluation metric, the best choice for confidence estimation is the Variance method. This makes intuitive sense since Variance measures the *spread* of the classifiers' scores: Higher the spread, more unambiguous (and hence, more confident) the ranking of the predicted labels is. On the other hand, when using Accuracy as the evaluation metric, the best choice seems to be the Min method, which puts a lower bound on the confidence of all classifiers involved, hence ensuring a minimum level of performance on the given task. Thus, we choose Variance and Min when optimizing the task orders with respect to MAP and F_1, respectively.

Interestingly, the correlation values vary significantly with the choice of the performance metric, which shows that MAP and Accuracy have sufficiently different behavior. In fact, MAP and Accuracy were found to be only moderately correlated ($\rho = 0.4582$) with each other on this dataset.

[2] Accuracy can be computed individually for each prediction task on each document. Therefore we use it in this experiment as a substitute for F_1, which is an aggregate metric computed over the entire dataset instead of per-document

Table 5: Correlation between various confidence estimation methods and two evaluation metrics

	Correlation with	
Method	MAP	Accuracy
`Mean`	0.1696	0.1850
`GeoMean`	0.1784	0.2420
`Min`	0.1981	**0.2740**
`Max`	0.0988	0.1679
`Variance`	**0.2440**	−0.0087

5.3 Effect of Task Order on Prediction Performance

It is important to understand why and how the order of the prediction tasks affects their overall performance, as this may not be obvious at first. In our setup, any task ordering is deemed to put certain tasks at an advantage by placing them later in the task order, which will give them more data for prediction, at the expense of other tasks that are placed early on and therefore would have less data for prediction. Since all tasks of each document must be predicted by the system eventually, it may not be immediately clear why the prediction order should affect the *overall* performance, because these advantages and disadvantages incurred by prediction tasks should balance each other out.

In more formal terms, information gain is symmetric, i.e., given two random variables x_i and x_j, the reduction in entropy (uncertainty) of x_j when x_i is known is exactly equal to the reduction in entropy of x_i when x_j is known. Therefore, in our setup, the task orders T_iT_j and T_jT_i should – in theory – perform equally well because any gain in performance on task T_j obtained by placing it after T_i should be balanced by the gain in performance on T_i in the latter task order. However, we show that this does not hold true in practice. The relationship between entropy and classifier performance is not constant for all target variables: A certain reduction in entropy of two variables does not necessarily lead to the same improvement in their corresponding predictive performance, due to the nature of the decision surface for each prediction task as well as the inductive bias of the prediction models used.

We conducted a controlled experiment where we only flip two adjacent tasks while holding the order of the other tasks constant (i.e., disabling Φ in Algorithm 1). Note that only the performance on this pair of tasks will be affected by the flip: In a given task order, all tasks that appear before this pair cannot be affected since this pair is simply absent during their prediction. For all subsequent tasks, this pair is present in a *bag* of features; the order in which they were obtained is irrelevant at later prediction steps.

We use the Accenture dataset for its larger size and choose the prediction `PertinentToOrgUnit` (T_3) and `Keywords` (T_4) as the two tasks, each admitting 20 distinct values (see Table 1)[3]. An experiment comprises two runs: (i) processing the documents with the order ...T_3T_4..., and (ii) with the order ...T_4T_3...[4]. For each experiment, we can also choose different adjacent positions for these two tasks; we try three alternatives – positions 1–2, 5–6 and 10–11, i.e. at the beginning, middle and end of the total ordering of the tasks. In each experiment, we measure the average performance (in terms of MAP) on T_3 and T_4 for the two orders, as shown here:

Order	MAP(T_3)	MAP(T_4)
...T_3T_4...	a	b
...T_4T_3...	c	d

Not surprisingly, $b \geq d$ and $c \geq a$ in all experiments, since in each ordering the latter prediction task has more data to work with. However, we are interested in knowing whether $a + b = c + d$, in which case the order of these two tasks will not affect the overall performance of the system. Equivalently, we want to know if $b - d = a - c$, i.e., whether the performance gained on T_4 by placing it after T_3 is balanced by a corresponding loss in T_3's performance. Interestingly, this is not the case. We plot the corresponding performance gain and loss on these tasks when they are placed at various adjacent positions in the ranked list (Table 6). Thus, even though the decrease in entropy of one prediction task is offset by the same amount of increase in the other for the two orders, this does not translate into equal performance gain and loss on the two prediction tasks – i.e., one of the orders (T_3T_4 in this case) is more favored than the other and leads to higher overall performance. These observations provide an empirical justification for directly taking into account the classifier's behavior as well as the performance metric, instead of measuring quantities like entropy or information gain that are only indirectly related to the classifier's performance on a given task.

Moreover, when we place the pair at latter positions (e.g., "10–11") in the total task order, both the gain as well as corresponding loss diminish. This is to be expected, since more task outcomes are available at latter positions for making the predictions for both the tasks, introducing redundancy of information, and

[3] We tried other task pairs, with similar results.

[4] The ordering of the rest of the tasks is irrelevant for this experiment.

Table 6: Effect on performance (MAP) due to flipping two adjacent tasks, T_3 and T_4. The performance gain $(b-d)$ on T_4 is not exactly canceled by loss $(a-c)$ on T_3 at any of the pairs of adjacent positions in the task order, leading to a non-zero change in overall performance ($\Delta = (b-d) + (a-c)$).

Positions	$(b-d)$	$(a-c)$	Δ
1–2	+0.2339	-0.0360	+0.1979
5–6	+0.0628	-0.0124	+0.0504
10–11	+0.0218	-0.0046	+0.0172

therefore, making the particular orientation of the pair less crucial. In other words, when available data is sparse, the order of predictions has a much more dramatic effect on the overall performance.

6 Discussion

In this paper, we have considered a specific sequential prediction setting consisting of multi-attribute documents. However, our proposed approach is more generally applicable to any setting where the system must determine the order in which user feedback should be solicited, but at the same time, the system must also exhibit good predictive performance on the items selected for user feedback, thus requiring a trade-off between system benefit and user benefit. Such scenarios arise naturally when active learning must be performed during test phase or prediction phase, when the system interacts with the *end-user*, who expects good system performance, instead of interacting with an *oracle*, who merely provides correct labels as demanded by the system. A notable example is the exploration-exploitation problem in the context of adaptive filtering, where the information filtering system must decide whether to present a document to the user based on two potentially conflicting criteria: The expected utility of the document to the user, and the expected utility of the user's feedback to the system [30]. An important part of our future work in this direction is the application of these ideas to other mainstream machine learning problems like collaborative filtering.

In our experiments, we have only considered classification problems (Section 4.3) as the prediction tasks. However, our approach provides a template that can accommodate any combination of classification, regression, retrieval, or ranking systems, as long as the systems can provide a measure of the degree of confidence (or uncertainty) of their predictions. A lot of research has focused on estimating and calibrating the confidence scores of various learning algorithms [9, 14, 15, 24], including the ones that involve structured outputs [8, 10, 22], since confidence estimation forms the basis for uncertainty sampling based approaches to active learning [7, 15].

7 Acknowledgements

This work is supported in parts by the National Science Foundation (NSF) under grant IIS_0704689, and fellowships from Yahoo! Inc. and Accenture Inc. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8 Conclusions

We have proposed an *active task ordering* strategy for dynamically ordering a series of prediction tasks to optimize their overall performance. We take into account the effect of user feedback on the prediction confidence of the systems in the context of the current input instance to determine the best prediction order. Our experiments on two datasets show that the proposed approach is effective and also a computationally efficient and online alternative to our previously proposed static ordering strategy, which requires additional data for offline computation of the task order and relies on the solution to an NP-hard problem. Moreover, we evaluate the correlation between various formulations of classifier confidence for multi-label classification problems and the true prediction performance, highlighting the different behavior of two evaluation metrics. The proposed approach represents a general strategy that is applicable to any setting where a trade-off between system benefit and user benefit is required, and can accommodate any combination of classification, regression, retrieval, and ranking systems as long as their prediction confidence can be estimated on a common scale.

References

[1] C. Buckley and E. Voorhees. Retrieval system evaluation. *TREC: Experiment and Evaluation in Information Retrieval*, pages 53–75, 2005.

[2] R. Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

[3] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM New York, NY, USA, 2006.

[4] C. Chang and C. Lin. LIBSVM: A library for support vector machines. *Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm*, 80:604–611, 2001.

[5] I. Charon and O. Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR: A Quarterly Journal of Operations Research*, 5(1):5–60, 2007.

[6] J. Chen and J. Shao. Nearest neighbor imputation for survey data. *Journal of Official Statistics, Stockholm*, 16(2):113–132, 2000.

[7] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[8] A. Culotta and A. McCallum. Reducing Labeling Effort for Structured Prediction Tasks, 2005.

[9] S. Gandrabur and G. Foster. Confidence estimation for text prediction. In *Proceedings of the Conference on Natural Language Learning (CoNLL 2003)*, 2003.

[10] R. Hwa. Sample Selection for Statistical Parsing. *Computational Linguistics*, 30(3):253–276, 2004.

[11] T. Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.* Springer, 1997.

[12] A. Lad, Y. Yang, R. Ghani, and B. Kisiel. Toward optimal ordering of prediction tasks. In *SDM*, pages 884–893. SIAM, 2009.

[13] D. Lewis. Evaluating text categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 312–318. Morgan Kaufmann, 1991.

[14] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156, 1994.

[15] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc. New York, NY, USA, 1994.

[16] H. Lin, C. Lin, and R. Weng. A note on Platt's probabilistic outputs for support vector machines:[Technical report]. *Department of Computer Science and Information Engineering, National Taiwan University*, 2003.

[17] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and computation*, 108:212–212, 1994.

[18] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 1999.

[19] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Proc. of 20th European Conference on Machine Learning*, 2009.

[20] G. Reinelt. The linear ordering problem: algorithms and applications, Research and Exposition in Mathematics 8. *Berlin: Heldermann*, 1985.

[21] R. Rifkin and A. Klautau. In Defense of One-Vs-All Classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[22] C. Thompson, M. Califf, and R. Mooney. Active Learning for Natural Language Parsing and Information Extraction. In *Machine Learning-International Workshop then Conference*, pages 406–414. Morgan Kaufmann Publishers, Inc., 1999.

[23] S. Thrun. Is Learning the n-th Thing Any Easier Than Learning the First? *Advances in Neural Information Processing Systems*, pages 640–646, 1996.

[24] S. Tong. *Active Learning: Theory and Applications.* PhD thesis, Stanford University, 2001.

[25] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[26] I. Tsochantaridis and T. Hofmann. Support vector machines for polycategorical classification. *Lecture notes in computer science*, pages 456–467, 2002.

[27] C. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann Newton, MA, USA, 1979.

[28] Y. Yang. A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 137–145. ACM New York, NY, USA, 2001.

[29] J. Zhang, Z. Ghahramani, and Y. Yang. Learning Multiple Related Tasks using Latent Independent Component Analysis. *Advances in Neural Information Processing Systems*, 18:1585, 2006.

[30] Y. Zhang, W. Xu, and J. Callan. Exploration and Exploitation in Adaptive Filtering Based on Bayesian Active Learning. In *Machine Learning-International Workshop then Conference*, volume 20, page 896, 2003.