Toward Optimal Ordering of 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

Rayid Ghani
Accenture Technology Labs
rayid.ghani@accenture.com

Bryan Kisiel
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
bkisiel@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 performed may have a significant impact on the overall performance of the prediction systems. However, manual specification of an optimal order may be difficult when the interdependencies are complex, especially if the number of tasks is large, making exhaustive search intractable. This paper presents the first attempt at solving the optimal task ordering problem using an approximate formulation in terms of pairwise task order preferences, reducing the problem to the well-known Linear Ordering Problem. We propose two approaches for inducing the pairwise task order preferences – 1) a classifier-agnostic approach based on conditional entropy that determines the prediction tasks whose correct labels lead to the least uncertainty for the remaining predictions, and 2) a classifier-dependent approach that empirically determines which tasks are favored before others for better predictive performance. We apply the proposed solutions to two practical applications that involve computer-assisted trouble report generation and document annotation, respectively. In both applications, the user fills up a series of fields and at each step, the system is expected to provide useful suggestions, which comprise the prediction (i.e. classification and ranking) tasks. Our experiments show encouraging improvements in predictive performance, as compared to approaches that do not take task dependencies into account.

1 Introduction

Many information-centric applications involve a series of user interactions to collect data, usually in the form of records, where each record contains a set of fields. Such scenarios are ubiquitous – many Web pages on the Internet ask the user to fill up forms or questionnaires, which are per-user records with a fixed set of fields. Similarly, large organizations spend considerable efforts in collecting and managing knowledge, which generally involves systematically maintaining records of past business endeavors, and each such record takes the form of a set of fields like *problem statement*, *goals*, *deliverables*, *project category*, *people assigned*, *products used*, *outcome*, and so on. An information system that can successfully assist users in filling up such forms and reduce their burden in terms of time and effort has a tremendous potential to increase the volume as well as the quality of the collected data, while keeping the costs down.

One way in which an information system can assist users is by predicting the most appropriate values for the next field and suggest them to the user at each step in the sequential form-filling process. The prediction process is generally accomplished by using classification or retrieval systems that take the values of fields that have already been entered as input, and produce a list of candidates for the next field – i.e., the target field. We call each such classification or retrieval step as a *prediction task*. These prediction tasks can be accomplished using any of the state of the art classifiers like Support Vector Machines (SVM) [28], Logistic Regression [1], K Nearest Neighbors (KNN) [12], etc., or more recent approaches like Probabilistic Relational Models [15, 25] that jointly model the relationships between the fields.

In this paper, we investigate the problem from a slightly different angle – *Is there an order in which the prediction tasks should be posed, so as to maximize the overall accuracy of the predictions?* In other words, in-

stead of passively producing a response when presented with a target field, can the system itself choose the order in which the user interaction should proceed, so that it can maximize its predictive performance?

Consider a very simple example of form filling – Yahoo! Answers[1], a community-driven website that allows users to submit questions as well as answers to questions posted by others. To submit a new question, a user must first type in the question text, and then choose an appropriate category. One can think of *question text* and *category* as two fields that the user is required to fill in. Since *question text* is submitted first, the system assists the user in filling up the *category* field by presenting its top five suggestions. It is obvious that this order of presenting the two fields to the user allows the system to utilize a text classifier that takes the question text as input and produces a ranked list of categories. On the other hand, asking the user to input the category of the question before he or she enters the question text is unreasonable – the system would not be able to assist the user by presenting a small list of categories since it has no input to begin with, nor assist him or her in framing the question text because of the large space of possible questions even when the category is known. In this example with two fields, the asymmetric relationship between the fields is immediately obvious, making it easy to manually determine the *optimal ordering* of fields for better user experience.

However, there are many applications that involve a large number of fields with complex relationships and interdependencies, in which case the optimal order in which they should be presented to the user might not be obvious. One such example is trouble report generation for complicated problems, where each trouble report from the customer triggers a sequence of interdependent decisions, which correspond to filling up various fields in a report, like *problem summary*, *priority*, *category*, *sub-category*, *expert assigned*, *hardware type*, *software type*, and *resolution*. The order of some of the fields is fixed – e.g., a new report is created based on a *problem summary*, while *resolution* marks the end of the report. The order of other fields is usually arbitrary; for such fields, the system can provide a better ordering based on trouble-shooting behavior of users as well the performance of the prediction systems with respect to various orderings in the past data. For example, the system might suggest putting the *priority* field before the *engineer* field because historical data suggests that the latter decision depends heavily on the former. How to optimize the task order by simultaneously taking into

account all such dependencies is an open challenge.

Another example is document annotation for knowledge management. Consulting companies meticulously index documents from past projects and annotate them with many fields e.g. *abstract*, *topic*, *author*, *teams involved*, *offerings*, *alliances*, *client*, *domain specialty*, and with numerous controlled vocabularies like *industry keywords*, *technology keywords*, *vendor product keywords*, *pertinent to organizational unit*, *pertinent to service line*, *pertinent to country*, and so on. An effective information system can not only assist the user in annotating documents, but also in leveraging past data to select the most appropriate teams, experts, vendor products, and to determine which clients and countries to target when embarking on a new project. However, what statistical dependencies exists among these fields with respect to the given data may not be obvious even to experts who designed these taxonomies.

Our goal in this paper is to explore automated solutions to the problem of optimal task ordering that can be easily applied to any domain involving a set of prediction tasks that must be performed in a sequential manner through user interaction. Our main contributions are:

1. A formulation of optimal task ordering as an optimization problem with an explicit objective function that allows different performance criteria to be easily plugged in and compared. Specifically, we compare two main alternatives of modeling the pairwise interactions among tasks.

2. Reduction of an intractable optimization problem to the well known Linear Ordering Problem (LOP) [9], by relaxing the optimization criterion to partial order preferences over task pairs. This reduction allows us to leverage state of the art algorithmic solutions to the LOP problem for efficiently solving the task ordering problem.

3. Thorough evaluation of the proposed approaches on two datasets from important application domains. This is the first comparative evaluation of optimal task ordering approaches in the information retrieval and data mining domain to our knowledge.

In the next section, we formalize the optimal task ordering problem and state the associated challenges. In Section 3, we propose an approximate solution that breaks the problem into two manageable steps that are both amenable to practical solutions. We provide details of our experiments in Section 4, and present the results in Section 5.

---

[1]http://answers.yahoo.com

2 The Optimal Task Ordering Problem

Consider a set of n records or forms, and for each record, the user needs to fill in k fields, $\{F_1, F_2, ..., F_k\}$.[2] Additionally, let F_0 denote any data that are always available a-priori as input and should be treated as the starting point for filling up rest of the fields for each record, e.g., the *problem summary* in the case of trouble reports, and *abstract* and *title* in the case of document annotation. The system is expected to assist the user in filling up each of the k fields in a stepwise fashion, which corresponds to the k prediction tasks, $\{t_1, t_2, ..., t_k\}$, respectively.

Let π denote a bijection of $\{1, 2, ..., k\}$ onto itself, such that $t_{\pi(j)}$ denotes the jth task in the permutation induced by π. In the jth step of the prediction process, the system provides a prediction for the field $F_{\pi(j)}$, based on the values of the fields already visited by the user in the previous steps, $\{F_{\pi(1)}, F_{\pi(2)}, ..., F_{\pi(j-1)}\}$. This corresponds to the prediction task $t_{\pi(j)}$. If the predicted value for $F_{\pi(j)}$ is wrong, the user provides the correct value, otherwise he or she simply moves on to the next step.

Let the prediction performance in the jth step be denoted by $\Delta(t_{\pi(j)}^{(i)}, F_{\pi(j)}^{(i)})$, 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 recall, precision, F_β, MAP, MRR, etc., or some other appropriate metric that measures the utility of the system for the given domain.

The optimal task ordering problem can then be written as finding the permutation π^* that maximizes the average performance of tasks on the n records:

$$(2.1) \qquad \pi^* = \arg\max_{\pi} \ \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \Delta(t_{\pi(j)}^{(i)}, F_{\pi(j)}^{(i)})$$

where $t_{\pi(j)}^{(i)}$ denotes the jth prediction task in the ith record.

No efficient solutions are known for the formulation in equation 2.1. The difficulties in solving this formulation arise from the large size of the sample space, and also the large number of dependencies that needs to be considered – The jth prediction task $t_{\pi(j)}^{(i)}$ depends on all the preceding fields in the permutation, $\{F_{\pi(1)}, F_{\pi(2)}, ..., F_{\pi(j-1)}\}$.

A simple but inefficient approach is to try all permutations of the fields in the past data, run the classifier on all such permutations and select the permutation that gives the best average performance on the

---

dataset. However, such an exhaustive search for the best permutation is intractable since the number of possible permutations rises factorially with the number of fields, leading to $O(nk!)$ evaluations to find the optimal order. This is impractical for large datasets, and even small number of fields.

3 Proposed Approach

The above-mentioned formulation does not suggest any efficient way of exploring the large search space. In order to proceed, we propose an approximate solution that breaks the problem into two steps:

1. Instead of directly attempting to find the *total* order that will maximize the performance on the prediction tasks, we find *pairwise preferences* among the task orders.

2. Use the pairwise preferences to derive an optimal total ordering of the tasks.

We show that both these are steps can be accomplished practically for moderately sized problems, and are amenable to multiple alternative approaches.

3.1 Pairwise Preferences of Task Orders
Given k tasks, consider a $k \times k$ matrix Ω, such that Ω_{ij} denotes the *preference score* or *expected benefit* of placing task t_i before t_j in the total ordering, but not necessarily adjacent to each other. Once we have such preference scores for each pair of tasks, our objective is reduced to finding the total order that maximally satisfies the pairwise preferences.

Irrespective of how Ω is estimated, such an approach leads to an approximation of the problem since it only models the pairwise dependencies among tasks, while ignoring higher order interactions. For instance, Ω_{ij} simply represents the estimated benefit of placing task t_i before t_j, independent of what other tasks are placed before t_i, or between t_i and t_j. In other words, we estimate the pairwise preferences of tasks independent of their absolute ranks and the ranks of other tasks in the total ordering. On the other hand, this means we only need to estimate $O(n^2)$ entries instead of the modeling $O(2^n)$ potential interactions between tasks.

We now consider two main alternatives to estimating Ω – (i) a classifier-agnostic approach that captures the uncertainties of the fields in terms of their conditional entropies, and (ii) a classifier-dependent approach that empirically observes the performance of the classifier with respect to different pairwise task order constraints, and hence leads to task orders that take the behavior of the classifier into account. We explore these approaches in the following sections.

3.1.1 Classifier-Agnostic Approach One way to populate the Ω matrix is to use the conditional entropies of the fields as indicators of the pairwise preferences among prediction tasks. Treating each field as a discrete random variable, the conditional entropy of field F_j given F_i is defined as:

$$H(F_j|F_i) = \sum_{f_i} p(f_i) H(F_j|F_i = f_i)$$
$$= -\sum_{f_i} \sum_{f_j} p(f_i, f_j) \log p(f_j|f_i)$$

The expected benefit of performing t_i before t_j is represented by the negative entropy of F_j conditioned on F_i:

$$(3.2) \qquad \Omega_{ij} = -H(F_j|F_i)$$

Thus, a low entropy of F_j conditioned on F_i leads to a high preference for placing F_i before F_j.

A related metric is Information Gain (IG), which is defined as $IG(F_i, F_j) = H(F_j) - H(F_j|F_i)$. Note that IG is a symmetric measure with respect to F_i and F_j, and therefore does not enforce a directional preference between any pair of tasks. It represents the *reduction* in the entropy of any one of the fields given the other. However, we are not interested in the reduction of the entropy, but in the *conditional* entropy of the target field when the other field is observed, since that will have a more direct bearing on how well a classifier can perform on the target field.

Note that conditional entropies are independent of the choice of the classifier or the evaluation metric. Our ultimate objective to maximize the performance of the given classifier in terms of a given evaluation metric, not in terms of entropy. To understand whether an approach that takes the behavior of the classifier as well the evaluation metric into account can lead to better task orders, we present an alternative classifier-dependent approach, as described next.

3.1.2 Classifier-Dependent Approach Instead of using entropy as a surrogate for the true performance metric, another way of populating the Ω matrix is to use held-out data to directly observe the performance of the classifier under different pairwise task order constraints. Algorithm 1 describes the steps involved. We use a set of documents, D, with field orders randomly permuted per-document, and then set each Ω_{ij} as the average performance on those documents where t_i appeared before t_j, denoted by the set S_{ij}. Note that steps 1–4 represent a sampling of the random permutations, which is only limited by the computational resources. In particular, step 2 of the algorithm can be performed multiple times

---

Algorithm 1 Classifier-Dependent Approach for calculating Pairwise Preferences
___
Require: Data set D, trained classifier, performance metric Δ.
1: **for** each document d_n in D **do**
2:      Choose a random permutation π_n
3:      Run classifier on d_n using task sequence π_n, and record the performance Δ_n.
4: **end for**
5: **for** each pair (i, j) in k^2 **do**
6:      $S_{ij} \leftarrow \{n : \pi_n(i) < \pi_n(j)\}$
7:      $\Omega_{ij} \leftarrow \frac{1}{|S_{ij}|} \sum_{n \in S_{ij}} \Delta_n$
8: **end for**
9: **return** Ω
___

per document, and multiple corresponding values of π_n and Δ_n can be recorded and used accordingly in steps 5–8.

This approach is dependent on the classifier as well as the performance metric that we ultimately care about. Therefore it can naturally accommodate performance metrics that give different weights to each field, or different costs to various classification errors.

3.2 From Pairwise Preferences to Optimal Order Our goal is to use the preference matrix Ω (computed using one of the above-mentioned ways) to derive an optimal total order of the tasks π^*. A reasonable way to frame the problem is to find the total order that maximizes the sum of pairwise preferences:

$$(3.3) \qquad \pi^* = \arg\max_{\pi} \sum_{(i,j):\pi(i)<\pi(j)} \Omega_{ij}$$

Compare this with our earlier formulation (equation 2.1), which involved arbitrary dependencies among fields. In contrast, equation 3.3 only involves pairwise interactions of the elements of the permutation.

The solution to this formulation is already known – it is equivalent to the Linear Ordering Problem (LOP) [23]. However, LOP is known to be NP-Hard, but state of the art solvers can handle moderately sized data. We give an outline of the Linear Ordering Problem, with a brief survey of recent developments in algorithms and the sizes of the problems that they can handle practically.

3.2.1 Linear Ordering Problem The Linear Ordering Problem appears in literature under various names – Maximum Acyclic Subdigraph Problem, The Maximum Consistent Arc Set, or Median Ordering Problem and arises is various fields including social sci-

Table 1: Run-times of a state of the art LOP solver on graphs with equi-probable directions of edges

Matrix Size	CPU Time (secs)
15×15	2
24×24	8
30×30	50
33×33	800

Table 2: The Accenture Dataset

Field 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

ences, electrical engineering and mathematics [9, 3]. It is generally defined in terms of a special kind of directed graph called *Tournament*.

A weighted Tournament $T = (V, E, w)$ is a digraph such that for any pair of vertices v_1 and v_2 in V, there is one and only one edge (v_1, v_2) or (v_2, v_1) in E. Let the weight of an edge (v_1, v_2) be denoted by a positive weight function w_{12}. If $w_{12} < 0$, replace the edge (v_1, v_2) with the edge (v_2, v_1) of weight $w_{21} = -w_{12}$. A linear order π defined on V is an ordering of the vertices $v_{\pi(1)} > v_{\pi(2)} > ... > v_{\pi(n)}$. The Linear Ordering Problem finds an order π that has minimum *remoteness* with respect to T – i.e., the sum of weights of the edges in T that must be reversed to get the linear order π:

$$\pi^* = \arg \min_{\pi} \sum_{(i,j):\pi(i)>\pi(j)} w_{ji}$$

The problem is known to be NP-hard [14, 9]. However, considerable progress has been made in the last decade, steadily increasing the size of the problems that can be solved in reasonable amounts of time [8, 22]. For instance, Table 1 shows the runtimes obtained by a state of the art algorithm that uses a branch and bound search with various heuristics to improve the bounding function [9]. The running time of the algorithm greatly depends on the nature of the graph; we have only shown the runtimes on problems with equal probability of an edge (i, j) or (j, i), since this corresponds to the LOP problem that arises in our case.

To fit the standard LOP formulation, we must tweak our Ω matrix to have one and only one edge between each pair of vertices. We calculate the difference between each pair Ω_{ij} and Ω_{ji}:

$$\Omega' = \Omega - \Omega^T$$

and keep only the positive edge among Ω'_{ij} and Ω'_{ji}.

Various polynomial time approximate algorithms have also been proposed for LOP [4, 17, 18]. However, since our problem settings (cf. Section 4.1) only involve matrices smaller than 15×15, we restrict attention to

the exact algorithm due to [9], who have also made their implementation publicly available.[3]

4 Experiments

Our experimental goals are two-fold – 1) To compare the proposed approaches against baseline approaches that do not take task interdependencies into account, and 2) to compare and understand the behavior of the proposed approaches against each other with respect to various performance criteria.

To this end, we use two datasets, both in the form of records comprising multiple fields. Predicting the correct values for each field corresponds to the prediction tasks. The goal is to re-order these tasks so as to maximize the overall quality of the predictions produced by the system.

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 fields. *Abstract* and *Title* are treated as the initial fields (i.e., F_0, see section 2). We must make predictions for the rest of the 11 fields. Table 2 gives the details of the fields in 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 fields. *Problem summary* is treated as the initial field, and the rest of the 12 fields

Table 3: The F/A-18 Dataset

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

are predicted in a stepwise manner. Table 3 gives the details of the fields in 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 the pairwise preferences matrix Ω, as described in section 3.1.

3. **Test set** – This portion was used to evaluate the performance of the classifiers using the optimal task orders as learned using the Order training set.

We switch the roles of the three parts and present the mean and standard deviation of the respective performance on the 6 combinations, to get a better sense of the robustness of the algorithms.

4.2 Evaluation Metrics We evaluate and compare the proposed approaches in terms of the predictive performance of the classifiers under different task orders generated by the proposed approaches. We use three evaluation metrics that capture slightly different aspects of the system's performance – Mean Average Precision (MAP), F_1, and a custom-defined Utility metric.

MAP [5] is a rank-based metric that measures the quality of ranked list produced by a system. Given a set of n items to be ranked and a corresponding set of binary judgments $\mathbf{y} = \{y_1, y_2, ...y_n\}$, the Average Precision of a system-produced ranked list $\hat{\mathbf{y}} = \{\hat{y_1}, \hat{y_2}, ...\hat{y_n}\}$ is given by:

$$MAP(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{R} \sum_{j:\hat{y}_j=1} Prec@j$$

where R is the number of relevant items in the true judgments $\mathbf{y}$, and $Prec@j$ is the fraction of items in the top j ranks of the system-produced list $\hat{\mathbf{y}}$ that are relevant. MAP is computed as the mean of the Average Precision scores for each query – i.e., each field in our case.

$\mathbf{F_1}$ is a set-based metric, equal to the harmonic mean of Recall R (fraction of relevant items retrieved) and Precision P (fraction of retrieved items that are relevant) [27]:

$$F_1 = \frac{2RP}{R + P}$$

In the case of form filling, *relevant items* correspond to the correct item(s) for a field, as determined from past data with completely filled records; *retrieved items* correspond to all the items that were suggested by the system. 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 categories, while micro-average is dominated by performance on categories with many documents [20].

Utility-based metrics The above-mentioned metrics do not necessarily reflect the end-user experience in a form-filling interface. Instead, one might want to estimate the amount of effort expended by the user, with and without the aid of a prediction system. The effort can be estimated in terms of the number of additions and deletions that the user needs to make in each field, similar in spirit to *string edit distance*. Specifically, we define the user effort as:

$$E = 2 * |additions| + |deletions|$$

assuming that additions are twice as costly as deletions when correcting the system's suggestions.

It is usually harder to choose the correct value for a field that has many possible fillers, e.g. *Point Of Contact* in the F/A-18 dataset, which can take one of 34 values, as opposed to fields like *Operating System*, which only takes 4 different values. To reflect this intuition, we define a *weighted* version of user effort that penalizes additions more for fields that have a bigger vocabulary:

$$E_w = 2 * |additions| * \log(V) + |deletions|$$

where V is the size of the vocabulary for the given field.

Using these two definitions of user effort, we define the corresponding unweighted and weighted Utility metrics as:

$$(4.4) \qquad \text{UTILITY} = 1 - \frac{E}{E_0}$$

$$(4.5) \qquad \text{UTILITY-W} = 1 - \frac{E_w}{E_{w0}}$$

Table 4: **Accenture dataset** – Performance (columns) of various task ordering approaches (rows). In four of the five applicable cases, the best performance (bold entry in each column) is obtained when the performance and task order optimization criterion are matched.

Task ordering method	Evaluation Metric				
	MicroF1	MacroF1	MAP	UTILITY	UTILITY-W
(Proposed approaches)					
LOP-MicroF1	**0.6355 ± 0.01**	**0.4459 ± 0.00**	0.7684 ± 0.01	0.4324 ± 0.01	0.3783 ± 0.00
LOP-MacroF1	0.6229 ± 0.01	0.4391 ± 0.01	0.7647 ± 0.01	0.4372 ± 0.01	0.3840 ± 0.01
LOP-MAP	0.6022 ± 0.01	0.3666 ± 0.01	**0.7691 ± 0.00**	0.4358 ± 0.01	0.4418 ± 0.03
LOP-UTILITY	0.5693 ± 0.01	0.3216 ± 0.01	0.7466 ± 0.01	**0.4784 ± 0.01**	0.5230 ± 0.01
LOP-UTILITY-W	0.5767 ± 0.01	0.3197 ± 0.01	0.7458 ± 0.00	0.4760 ± 0.01	**0.5334 ± 0.01**
LOP-CondEntropy	0.5971 ± 0.01	0.4282 ± 0.00	0.7628 ± 0.01	0.4493 ± 0.01	0.4021 ± 0.01
(Baselines)					
DecEntropy	0.6208 ± 0.01	0.4165 ± 0.00	0.7623 ± 0.01	0.4381 ± 0.01	0.3826 ± 0.01
Expert	0.5765 ± 0.01	0.3159 ± 0.01	0.7465 ± 0.00	0.4585 ± 0.01	0.5095 ± 0.01
Random	0.5343 ± 0.04	0.3066 ± 0.03	0.7194 ± 0.01	0.3985 ± 0.03	0.3695 ± 0.04

where E_0 and E_{w0} denote the unweighted and weighted effort, respectively, expended by a user when the system makes no predictions and the user must insert all correct values manually. Thus, a Utility score of 0.6 means that the proposed ordering leads to a 60% reduction in user effort as compared to an unaided user.

4.3 Classifiers We choose Linear Support Vector Machines (SVMs) for all our experiments, which have been reported to produce state of the art performance on many classification problems [7] and can handle a large number of potentially redundant features, which is common in the domain of text classification [19].

One SVM classifier is trained per field, treating it as the target variable, while the rest of the fields are treated as the predictors. A score-based thresholding strategy with a five-fold cross-validation was used to convert the scores produced by the classifiers into decisions [29]. Since each field can take multiple values, the prediction task is a multi-label classification problem, which we address using a one-vs-all scheme [24].

5 Results

Table 4 and 5 show the results obtained using various task ordering approaches on the Accenture and F/A-18 datasets, respectively. The proposed approaches include two main alternatives – the conditional entropy based method (LOP-CondEntropy in the tables), and the classifier-dependent methods that can be trained using various optimization criteria (the rest of the LOP-* rows in the tables).

We include three baselines:

1. **Random** – Arbitrary task orderings,

2. **Expert** – A task order suggested by a domain expert in the case of the Accenture dataset, and

3. **DecEntropy** – A greedy approach that orders the tasks by decreasing entropies. The intuition is that the fields with the most uncertainty (as measured by the entropy of their label distribution in the training set) should be placed first so that they provide maximum information to the classifier to make predictions for the rest of the fields. The inverse relationship between entropy and classification performance on a target variable is the basis of many active learning strategies based on uncertainty sampling [11, 33] as well as feature selection strategies based on mutual information [13, 30].

On both datasets, the proposed approaches achieve the best performance with respect to all evaluation metrics. The average performance of the random task orderings is the worst, and also has the highest variance. The Expert task ordering in Table 4 does not perform well in terms of MicroF1, MacroF1, and MAP, but has competitive performance in terms of unweighted and weighted Utility.

The classifier-dependent approach performs better than the conditional entropy based approach, since the former directly uses the evaluation metric as part of the optimization criterion. Moreover, in almost all cases, the best performance is obtained when the same metric

Table 5: **F/A-18 dataset** – Performance (columns) of various task ordering approaches (rows). In all of the five applicable cases, the best performance (bold entry in each column) is obtained when the performance and task order optimization criterion are matched.

	Evaluation Metric				
Task ordering method	MicroF1	MacroF1	MAP	UTILITY	UTILITY-W
(Proposed approaches)					
LOP-MicroF1	**0.8550 ± 0.00**	0.5572 ± 0.01	0.9013 ± 0.00	0.7783 ± 0.00	0.8024 ± 0.00
LOP-MacroF1	0.8449 ± 0.00	**0.5701 ± 0.01**	0.9009 ± 0.00	0.7775 ± 0.00	0.7997 ± 0.00
LOP-MAP	0.8460 ± 0.00	0.5583 ± 0.01	**0.9032 ± 0.00**	0.7796 ± 0.00	0.8031 ± 0.00
LOP-UTILITY	0.8469 ± 0.00	0.5602 ± 0.00	0.9030 ± 0.00	**0.7823 ± 0.00**	0.8052 ± 0.00
LOP-UTILITY-W	0.8377 ± 0.00	0.5516 ± 0.01	0.8994 ± 0.00	0.7737 ± 0.01	**0.8057 ± 0.00**
LOP-CondEntropy	0.8397 ± 0.00	0.5604 ± 0.00	0.9002 ± 0.00	0.7786 ± 0.00	0.8007 ± 0.00
(Baselines)					
DecEntropy	0.8468 ± 0.00	0.5634 ± 0.00	0.8994 ± 0.00	0.7767 ± 0.00	0.7979 ± 0.00
Random	0.7983 ± 0.01	0.5352 ± 0.03	0.8990 ± 0.01	0.7473 ± 0.02	0.7399 ± 0.02

is used for task order optimization as well as evaluation. This suggests that different performance metrics indeed lead to different optimal task orders. The differences in the behavior of the metrics are more striking in the case of the Accenture dataset (Table 4) – high performance on MicroF1 and MacroF1 usually corresponds to lower UTILITY and UTILITY-W scores, and vice versa. The classifier-dependent approach provides an effective way to optimize the task order with respect to different utility functions that are most appropriate for a given domain. By directly observing the behavior of the classifier and the evaluation metric, this approach can derive an optimal order in the absence of any domain knowledge as well as understanding of the prediction systems.

5.1 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 4, and similarly for Table 5.

For each pair of ordering methods, we compare their performance on each field of each document in terms of binary decisions made by the classifier. The total number of such decisions is equal to $n \times \sum_{k=1}^{K} |F_k|$, where n is the number of documents and $|F_k|$ denotes the number of possible categories for field F_k, since the classifier must make a binary decision for each such category for each field in each document.

The two sample paired sign test counts the number of times one method produces a better classification decision than the other method on the same document-field-category triplet, and calculates the p-value as the probability of observing this (or a more extreme) count under the *null hypothesis* – i.e. the two methods have the same performance. Therefore, a small p-value indicates a strong evidence against the null hypothesis.

On the Accenture dataset (Table 4), differences in UTILITY scores greater than 0.006 were found to be highly statistically significant (p-value $<<$ 0.01). On the F/A-18 dataset (Table 5), differences in UTILITY scores greater than 0.003 were found to be highly statistically significant (p-value $<<$ 0.01).

6 Related Work

The problem of learning to order items arises in many domains. In the context of Information Retrieval (IR), the task of learning to rank documents or Web pages with respect to queries has received much attention in recent literature [26, 6, 31, 21]. The objective is to learn a function that maps query-document features to document scores, which induce a ranking of the documents. In our case, instead of mapping features to scores of items to be ranked, the items (i.e. the prediction tasks) themselves interact with each other, thus affecting the overall predictive performance of different task orders. Thus, do not *learn* to rank tasks; we only induce a one-time ranking based on the task interactions as observed from past data.

Cohen et al. [10] used a formulation similar to ours for the problem of ranking Web pages when user feedback is available in the form of pairwise preferences. However, their main focus was on finding a good linear combination of preference functions. The task of finding the optimal order was addressed by using a simple

greedy algorithm.

In the context of machine learning, *Active Learning* [11] focusses on the problem of designing learning algorithms that can choose the instances to be labeled by the user (or *oracle*) instead of passively using labeled data given to them. However, the objective is to improve the performance of the learning algorithm by querying for the least number of labels from the user. The system is not expected to provide an initial guess when querying for a label. This allows the successful application of strategies that query for the most uncertain label [11, 33]. In our problem setup, we must choose the next field of the document by balancing two conflicting objectives – the correct label for the field should be beneficial for predicting the rest of the fields, and the system must already be reasonably confident about making an initial guess for the field to be queried. Another crucial difference is that in *Active Learning*, the learner actively queries for labels during the *training* phase. In our case, the querying takes place during the *test* or classification phase. In this sense, our setup is more closely related to the concept of *Active Classifiers* [16] that are designed to work with missing values during classification. An active classifier can, at a cost, ask for some of the unspecified attributes before deciding on the class label. However, they still maintain a distinction between input attributes and class labels, whereas this distinction is lost in our problem setup – each field behaves as an input attribute or target variable at different steps of the sequential prediction process.

Another related research area is multi-task learning that deals with simultaneously learning good models for multiple prediction tasks [2, 32]. The focus is on learning a set of interrelated prediction targets by leveraging their structural similarities with respect to a common set of input features. In our case, we have a set of prediction targets, but there is no common set of input features; instead, the prediction targets themselves comprise the predictors for the other target variables due to the *sequential* nature of the interactive process. This setup has not received significant attention in the field of machine learning.

7   Discussion and Future Work

Our approach, by design, provides an approximate solution to the task ordering problem. The approximation arises due to limiting attention to only pairwise task order preferences, while ignoring any higher order interactions between the prediction tasks. For instance, task t_j might be highly dependent on task t_i, thus creating a strong preference for ordering t_i before t_j. However, this preference might be obviated by the presence of a third task t_k, which is highly predictive of t_j. Ignoring

such interactions may restrict the algorithm from fully exploring the search space. While this is a *bias* in the algorithm introduced intentionally for tractability, its effect on the (sub-)optimality of the solution is difficult to characterize.

Secondly, we only considered *static* task ordering in this paper – using past data, an optimal order is learned, which is then held fixed for future user interactions. A more effective approach would be *active* task ordering, where the system dynamically chooses the next field based on the current state of user interaction.

8   Conclusions

This paper presents the first attempt at understanding and solving the optimal task ordering problem. We formulate optimal task ordering as an optimization problem with an explicit objective function that allows different performance criteria to be plugged in and compared. We reduce the intractable optimization problem to the well-known Linear Ordering Problem by relaxing the objective function in terms of pairwise task order preferences. We demonstrate how the proposed approach can be used to directly optimize the task order with respect to a utility metric that is appropriate for a given domain. Our results indicate encouraging improvements over baseline approaches that do not leverage task dependencies. We hope that this paper serves as a starting point for more sophisticated approaches that have a strong potential for improving user experience in a wide range of multi-step interactive processes.

9   Acknowledgements

References

[1] A. Agresti. *Categorical data analysis.* Wiley New York, 1990.

[2] R.K. Ando and T. Zhang. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *The Journal of Machine Learning Research*, 6:1817–1853, 2005.

[3] J.P. Barthélemy, B. Leclerc, and B. Monjardet. On the use of ordered sets in problems of comparison and consensus of classifications. *Journal of Classification*, 3(2):187–224, 1986.

[4] B. Berger and P.W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 236–243. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1990.

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

[6] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*. MIT Press, 2007.

[7] 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.

[8] I. Charon and O. Hudry. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097–2116, 2006.

[9] 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.

[10] W.W. Cohen, R.E. Schapire, and Y. Singer. Learning to order things. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10 table of contents*, pages 451–457. MIT Press Cambridge, MA, USA, 1998.

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

[12] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[13] G. Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of Machine Learning Research*, 3:1289–1305, 2003.

[14] M.R. Garey, D.S. Johnson, et al. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman San Francisco, 1979.

[15] L. Getoor, D. Koller, B. Taskar, and N. Friedman. Learning statistical models from relational data. In *Proc. AAAI*, pages 580–587, 2000.

[16] R. Greiner, A.J. Grove, and D. Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174, 2002.

[17] M. GrÖtschel, M. JÜnger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984.

[18] R. Hassin and S. Rubinstein. Approximations for the Maximum Acyclic Subgraph Problem. *Information Processing Letters*, 51(3):133–140, 1994.

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

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

[21] P. Li, C. Burges, Q. Wu, JC Platt, D. Koller, Y. Singer, and S. Roweis. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. *Advances in Neural Information Processing Systems*, 2007.

[22] J.E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. *High Performance Optimization*, pages 349–366, 2000.

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

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

[25] B. Taskar, E. Segal, and D. Koller. Probabilistic Classification and Clustering in Relational Data. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 870–878, 2001.

[26] M. Taylor, J. Guiver, S. Robertson, and T. Minka. SoftRank: optimizing non-smooth rank metrics. In *Proceedings of the international conference on Web search and web data mining*, pages 77–86. ACM New York, NY, USA, 2008.

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

[28] V.N. Vapnik. *Statistical learning theory*. John Wiley & Sons, 1997.

[29] 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.

[30] Y. Yang and J.O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *International Conference on Machine Learning*, pages 412–420. Morgan Kauffman Publications, INC., 1997.

[31] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278. ACM Press New York, NY, USA, 2007.

[32] 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.

[33] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields. *ICML 2003 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.