Multi-Task Learning for Spoken Language Understanding with Shared Slots

Xiao Li[1]*, Ye-Yi Wang*[1] *and Gokhan Tur*[2]

Microsoft Corporation
[1]One Microsoft Way, Redmond, WA 98052
[2]1310 Villa Street, Mountain View, CA 94041
{xiaol,yeyiwang,gokhant}@microsoft.com

Abstract

This paper addresses the problem of learning multiple spoken language understanding (SLU) tasks that have overlapping sets of slots. In such a scenario, it is possible to achieve better slot filling performance by learning multiple tasks simultaneously, as opposed to learning them independently. We focus on presenting a number of simple multi-task learning algorithms for slot filling systems based on semi-Markov CRFs, assuming the knowledge of shared slots. Furthermore, we discuss an intra-domain clustering method that automatically discovers shared slots from training data. The effectiveness of our proposed approaches is demonstrated in an SLU application that involves three different yet related tasks.

Index Terms: spoken language understanding, slot filling, multi-task learning

1. Introduction

Spoken language understanding (SLU) systems have traditionally focused on narrowly restricted domains. In the past a few years, the burgeoning development of voice search technologies opens up opportunities for developing web-scale SLU systems. For example, *Siri* [1] enables SLU for a dozen task domains such as *restaurants*, *movies* and *weather*. A similar application has been deployed in *Bing* for iPhone [2]. Since each task defines a different set of slots, one needs to have tailored training data for a supervised slot filling system.

Often times, one may find semantic correspondence between slots defined in different tasks. Consider these examples:

- Find an [*price*expensive] [*cuisine*italian] restaurant in [*location*new york city].
- Showtimes of [*title*harry potter and the deathly hallows] near [*place*redmond washington]

The first query requests a restaurant search, and the second asks about movie showtimes. We can see that *location* in the first task and *place* in the second task represent a similar semantic concept, although they are named differently. We let a *shared slot* denote a semantic concept that occurs in multiple tasks. When building multiple SLU systems with shared slots, it is possible that we achieve better performance by leveraging such information, as opposed to learning each system individually using task-specific training data.

Multi-task learning (MTL), originally introduced by [3] for performing multiple classification tasks jointly using neural networks, has been applied to various models and problems. In the area of SLU, [4] studied MTL for the problem of intent classification where labeled data is reused across tasks. In this work, we exploits MTL for *slot filling* and in particular for systems based on semi-Markov CRFs. Perhaps the closest work to ours is [5] which transfers information between linear-chain CRFs. Their work, however, does not explicitly define correspondence between CRF slots. Instead, they train a cascade of CRF model independently, using the prediction of the previous model as a feature, and perform joint decoding by a composition of CRFs. In this paper, in contrast, we first assume that the shared slots are given, and presents four MTL approaches that explicitly leverage such information. We then describe a clustering algorithm that automatically discovers shared slots from training data.

2. Semi-Markov CRFs

We start by providing a background of semi-Markov CRFs [6], upon which our slot filling models are constructed. Semi-Markov CRFs jointly model the segmentation of an input sequence and the classification of the segments. This enables the use of segment-level features, which brings great convenience and flexibility in feature engineering. Many works have shown the effectiveness of this model in information extraction from text [6, 7, 8] and in SLU [9, 10].

We let Y denote the set of labels (slots) for a given task. Given an input sequence $\mathbf{x}$, our goal is to obtain $\mathbf{s} = (s_1, s_2, \ldots, s_N)$, where each segment s_i is represented by a tuple (u_i, v_i, y_i). Here u_i and v_i are the indices of the starting and ending word tokens respectively; and $y_i \in Y$ is a segment label. We further augment the segment sequence with two special segments: (0, 0, *Start*) and (N+1, N+1, *End*), representing s_0 and s_{N+1} respectively. We further let $f(i, \mathbf{x}, \mathbf{s})$ denote a vector of feature functions, which takes on the form $f(\mathbf{x}, u_i, v_i, y_i, y_{i-1})$. There are usually two subtypes of feature functions, $f(y_{i-1}, y_i)$ and $f(\mathbf{x}, u_i, v_i, y_i)$. While the former solely depends on the label set Y, the latter is usually defined based on data observations. We will discuss the features used in our task in Section 5.2. Furthermore, we define $F(\mathbf{x}, \mathbf{s}) = \sum_{i=1}^{|\mathbf{s}|} f(i, \mathbf{x}, \mathbf{s})$. Similar to CRFs, semi-Markov CRFs are parameterized by a weight vector λ, each dimension corresponding to a feature function in f (and hence in F). Given training data, λ is estimated to maximize

$$p(\mathbf{s}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \lambda \cdot F(\mathbf{x}, \mathbf{s}) \quad (1)$$

where $Z(\mathbf{x}) = \sum_{\mathbf{s}' \in S_{|\mathbf{x}|,Y}} \exp \lambda \cdot F(\mathbf{x}, \mathbf{s}')$ is a partition function that sums over all possible segmentations and all possible labelings of the segments. Apparently the hypothesis space S depends on the input length $|\mathbf{x}|$ and the label set Y.

3. Multi-Task Learning Algorithms

When learning multiple tasks, $k = 1, \ldots, K$, we let Y_k and f_k (and hence F_k) represent the task-specific label set and feature function set respectively. For each k, we can estimate λ_k following a single-task learning scheme, *i.e.*, to maximize Equation (1) using task-dependent training data only.

If there are slots from different tasks representing a similar semantic concept, it is reasonable to believe that we can transfer information across tasks to improve the overall slot filling performance. In this section, we assume that the slot sharing information is given, which is represented by pairwise intersections $Y_k \bigcap Y_l$, and propose four simple MTL algorithms. Notice that in all approaches we assume the knowledge of the task ID of each training/test sample.

3.1. Soft evidence (SE)

We define $Y^0 = \bigcup_{k,l}(Y_k \bigcap Y_l)$ which represents the entire set of shared slots including the special slot *Other*. Our first approach constructs a global auxiliary model on Y^0, and then uses its prediction as SE in each slot filling system.

Specifically, we first learn λ_k, $k = 1, \ldots, K$, independently based on Equation (1) using their respective training data. As a separate process, we convert the labels of training data based on the following rule: a label is kept as it is if belonging to Y^0 and is otherwise converted to *Other*. In this way, the training data in all tasks will now have exactly the same label set, *i.e.*, Y^0. An auxiliary model is then learned using the converted training data from all tasks, which will presumably provide more reliable detection of the shared slots.

At decode time, we run an input sequence $\mathbf{x}$ through the auxiliary model, producing a segment sequence $\mathbf{e} = (e_1, \ldots, e_M)$, where $\mathbf{e} \in S_{|\mathbf{x}|,Y^0}$. Similar to the use of SE in CRFs [11], we condition $\mathbf{s}$ on two observation, $\mathbf{x}$ and $\mathbf{e}$, as well as the task ID k:

$$p(\mathbf{s}|\mathbf{x}, \mathbf{e}, k) \propto \exp\{\lambda_k \cdot F_k(\mathbf{x}, \mathbf{s}) + \omega \sum_{i=1}^{|\mathbf{s}|} \sum_{j=1}^{|\mathbf{e}|} \mathrm{I}(s_i = e_j)\} \quad (2)$$

where ω is a positive scalar controlling the importance of the SE. This model favors a hypothesis that is consistent with the prediction of the auxiliary model (w.r.t. both segment boundaries and labels). When $\omega = 0$, Equation (2) is equivalent to Equation (1). When $\omega = +\infty$, on the other hand, $\mathbf{e}$ becomes hard evidence, meaning that we fully trust the decision made by the auxiliary model.

3.2. Parameter tying (PT)

When there are shared slots, it is likely that the feature sets f_k have intersections as well. The PT approach still aims at learning multiple models λ_k, but the weights of the same feature functions are tied across tasks. PT can be considered as a form of regularization on model parameters.

Using an optimization scheme based on stochastic gradient descent, PT can be efficiently implemented by jointly estimating all task-dependent models. Specifically, we merge all training data and randomize the sample order. For a training sample from task k, we compute the gradient of λ_k based on Equation (1). When updating element in λ_k, we simultaneously update its counterpart elements in λ_l, $l \neq k$, if available.

3.3. Joint model (JM)

Alternatively, we learn a single joint model on the union set $Y^u = \bigcup_k Y_k$. The set of feature functions is also a union of task-dependent ones, which we denote as F^u; and we use λ^u to represent the corresponding weight vector.

The joint model λ^u is estimated from training data from all tasks, with the constraint that transitions can only occur between slots of the same task. In other words, we have

$$p(\mathbf{s}|\mathbf{x}, k) = \frac{1}{Z(\mathbf{x}, k)} \exp \lambda^u \cdot F^u(\mathbf{x}, \mathbf{s}) \quad (3)$$

where $Z(\mathbf{x}, k) = \sum_{\mathbf{s}' \in S_{|\mathbf{x}|,Y_k}} \exp \lambda^u \cdot F^u(\mathbf{x}, \mathbf{s}')$. At decode time, we use $\lambda^u \cdot F^u(\mathbf{x}, \mathbf{s})$ as a single decision function to search for the most likely segment sequence in each task.

The JM approach can be advantageous over PT in that it introduces new features to each task. Both approaches, however, are computationally expensive, which may not be feasible in practice when K is large.

3.4. Auxiliary data (AD)

In our last approach, we build task-dependent models using their own training data plus segments with "in-domain" labels from other tasks.

Consider k as the task of interest. For a training sample $(\mathbf{x}, \mathbf{s})$ from any task, we extract all segment subsequences $\bar{\mathbf{s}}$, where all labels in $\bar{\mathbf{s}}$ belong to Y_k and where the labels immediately preceeding and following $\bar{\mathbf{s}}$ do not belong to Y_k. In other words, $\bar{\mathbf{s}}$ is the locally longest subsequence with "in-domain" labels. $\mathbf{s}$ may have multiple such subsequences which form a set, which we denote as $\mathrm{sub}(\mathbf{s})$. Apparently if $(\mathbf{x}, \mathbf{s})$ comes from task k, $\mathrm{sub}(\mathbf{s})$ will contain one single sequence which is $\mathbf{s}$ itself. Given all such segments, we extract additional features that are absent from f_k. We let f_k^u denote the union of f_k and the set of new features. For task k, we estimate the corresponding weight vector λ_k^u by maximizing

$$\frac{1}{Z(\mathbf{x}, k)} \prod_{\bar{\mathbf{s}} \in \mathrm{sub}(\mathbf{s})} \exp \lambda_k^u \cdot F_k^u(\mathbf{x}, \bar{\mathbf{s}}) \quad (4)$$

where $Z(\mathbf{x}, k) = \sum_{\mathbf{s}' \in S_{|\bar{\mathbf{s}}|,Y_k}} \exp \lambda_k^u \cdot F^u(\mathbf{x}, \mathbf{s}')$. We can alternatively view this learning scheme as optimizing Equation (1) on task-specific training data, and optimizing Equation (4) on segments with "in-domain" labels from other tasks. At decode time, $\lambda_k^u \cdot F_k^u(\mathbf{x}, \mathbf{s})$ is used as the decision function for task k.

Unlike PT and JM that require learning multiple tasks all at once, AD decouples the optimization of task-dependent models and is hence scalable to a large number of tasks.

4. Discovering Shared Slots

Up to this point, we have assumed that the slot sharing information is given; and we yet need to discuss how to identify the intersections between $Y_1, \ldots, Y_K$. In other words, starting with K mutually exclusive sets of slots, our goal is to find intra-domain clusters; here by an "intra-domain cluster" we mean that no two elements in a cluster are from the same task domain.

We first construct a vector space representation for each slot. The dimensions of the vector space correspond to the feature functions of the form $f(\mathbf{x}, u_i, v_i, \cdot)$ (y_i is left out). The coordinates of a slot y, denoted by $\lambda(y)$, is determined by the weights associated with $f(\mathbf{x}, u_i, v_i, y_i = y)$. We learn the weights in a single-task learning setting following Equation (1).

In this way, $\lambda(y)$ naturally reflects the importance of each feature dimension in slot filling.

With this vector space representation, our next step is to group the slots into intra-domain clusters. To this end, we construct a K-partite view where the slots are partitioned based on their task IDs. We use $\mathcal{N}(y)$ to denote the set of neighbors of y, *i.e.*, slots in other tasks that are connected with y. $\mathcal{N}(y)$ are discovered incrementally with the following agglomerative clustering algorithm. At the end of the algorithm, each complete graph represents an intra-domain cluster, from which the intersections of $Y_1, \ldots, Y_K$, can be derived.

Algorithm 1 Intra-domain Agglomerative Clustering (IAC)

Input: $Y_1, \ldots, Y_K$, and $\lambda(y)$ for all $y \in \bigcup_k Y_k$

1: Initialize $\mathcal{N}(y) = \emptyset$ for all y;
2: For each pair $k \neq l$, compute the cosine similarity $d(y^1, y^2)$, $\forall y^1 \in Y_k$ and $\forall y^2 \in Y_l$;
3: **repeat**
4: Find (y^1, y^2) for which $d(y^1, y^2)$ is the largest and y^1 and y^2 are not connected;
5: **if** $\forall y \in \mathcal{N}(y^1)$, domain$(y) \neq$ domain(y^2); and $\forall y \in \mathcal{N}(y^2)$, domain$(y) \neq$ domain(y^1) **then**
6: Connect y^1 and y^2;
7: **end if**
8: **until** $d(y^1, y^2)$ is smaller than a threshold

Output: all complete graphs in this K-partite graph;

5. Evaluation

5.1. Data

We conducted experiments on three tasks. The data set consists of natural language queries in the domains of *Hotel*, *Restaurant* and *Movie*, *e.g.*, "book a hotel room for two in denver for next friday", or "best action movies 2011". We asked human annotators to manually label all three datasets at the slot level using labels defined in Table 2. We allocated 80% of each dataset for training and 20% for testing, with specific numbers of sentences and segments given in Table 5.1. Parameter estimation was done using 4-fold cross-validation on the training set.

	Training set		Test set	
Domain	Sent.	Slot	Sent.	Slot.
Hotel	1572	3729	406	992
Restaurant	2340	5269	601	1331
Movie	1768	2257	540	654

Table 1: Amounts of training/test data in each domain.

Furthermore, we let a human annotator to identify shared slots based on schema descriptions of all tasks and based on the training data. The top and middle sections of Table 3 show the human-identified shared slots.

5.2. Features

We implemented common features used in semi-Markov CRFs (see [8] for details). They include transition features, which are binary features defined on pair-wise segment labels. Lexical features used in our experiments consist of segment-level tokens, within-segment n-grams, preceding and succeeding n-grams, each in conjunction with possible segment labels. Furthermore, we incorporated features based on gazetteers such as lists of hotel names, restaurant names, movie titles, actor names and etc. The features include the match of segment-level tokens against gazetteers, as well as the match of preceding and succeeding tokens against gazetteers, again each paired with possible segment labels. We also used regular expression features to represent expressions of dates, times, quantities and monetary values. Finally, we implemented what we call combined features. One example feature can be that the current segment matches the regular expression for quantities while it preceeds the word "tickets", and the segment label is m-num-tickets (which presumably will have a positive weight). Such combined features are manually specified.

As a result, we have 300K-400K features in each domain. The large dimensionalities are largely due to the use of lexical features, which are extracted from training data, and due to the fact that we have conjunctions of each lexical feature with all possible segment labels. We did not apply any feature selection techniques as it is beyond the scope of our work. To avoid overfitting we sued l_2 regularization on model parameters.

Shared slots	**Original names**
location	h-location, r-location, m-location
near	h-near, r-near, m-near
date	h-checkin-date, r-reserve-date, m-reserve-date
time	r-reserve-time, m-reserve-time
price-range	h-price-range, r-price-range
star-rating	r-star-rating, h-star-rating
description	h-description, r-description
num-people	h-num-adults, r-party-size

Table 3: Top and middle sections: shared slots chosen by human annotator; Middle and bottom sections: shared slots discovered by IAC ranked by the order in which they are discovered.

		BS	SE	PT	JM	AD
Hotel	P	0.866	0.873	0.878	0.889	0.882
	R	0.873	0.882	0.874	0.887	0.882
	F1	0.870	0.878	0.876	**0.888**	0.882
Restaurant	P	0.859	0.863	0.878	0.874	0.879
	R	0.881	0.887	0.874	0.883	0.880
	F1	0.870	0.875	0.876	**0.879**	**0.880**
Movie	P	0.788	0.791	0.773	0.783	0.822
	R	0.769	0.771	0.803	0.815	0.783
	F1	0.779	0.781	0.788	**0.799**	**0.802**

Table 4: Comparision of MTL algorithms. Bold numbers indicate the best F1's that are statistically significant over baselines.

5.3. Results

The evaluation metrics include precision, recall and F1, all at the slot level, excluding the special slots *Other*, *Start* and *End*. A decoded segment is considered correct if and only if it has correct boundaries and has the same label as annotated. We used the same baselines as reported in our previous work [10][1], corresponding to a single-task learning setting. The results are given in the first column of Table 4.

[1]The numbers are slightly compared to [10] in a few cases due to the use of different learning rates

Domain	Label set
Hotel	h-amenities, h-checkin-date, h-checkout-date, h-description, h-hotel-type, h-location, h-name, h-near, h-num-adults, h-num-children, h-num-nights, h-num-rooms, h-price-range, h-reward-program, h-room-type, h-smoking, h-star-rating
Restaurant	r-amenities, r-cuisine, r-description, r-location, r-meal-type, r-menu, r-name, r-near, r-party-size, r-open-hour, r-price-range, r-restaurant-type, r-reserve-date, r-reserve-time, r-star-rating
Movie	m-actor, m-award, m-character, m-director, m-genre, m-language, m-location, m-movie-type, m-mpaa-rating, m-nationality, m-near, m-num-tickets, m-release-date, m-reserve-date, m-reserve-time, m-viewer-rating, m-title, m-theater

Table 2: Slot sets. Each set additionally includes an *Other* label, as well as two nominal labels *Start* and *End*. For readers' convenience, we named the slots such that the prefixes h-, r- and m- indicate labels in *Hotel*, *Restaurant* and *Movie* domains respectively.

To experiment with the proposed MTL algorithms, we first used the shared slots specified by the human annotator. We implemented the SE approach in Section 3.1. The auxiliary model was built on 5680 training-set queries that contain 4048 segments with shared slots (the remaining word tokens were assigned the *Other* label). We report in Table 4 the test-set performance with $\omega = 2$ which is found via cross-validation. Despite consistent improvements across tasks, the gains are not statistically significant in each individual task. Similar performance gains were observed in the PE approach.

In another set of experiments, we learned a single joint model λ^u as proposed in Section 3.3. The union feature set F^u contains nearly 2M features. By estimating the model λ^u on all training data, we obtained significant improvements over the baselines in all tasks. Moreover, by comparing with PT, we can see that using a universal feature set F^u brings great benefit to learning.

Finally, we experimented with the AD approach. The performance is comparable to the JM approach with a substantial reduction in computational complexity. Similar to the comparison between PT and JM, it is interesting to see how much gain comes from a better estimation of λ_k, and how much of that comes from adding new features. We thus ran another experiment by replace f_k^u and λ_k^u with f_k and λ_k respectively in Equation (4). The performance degraded but still outperformed the baseline. This again indicates that adding new data and new features are both beneficial.

5.4. Automatic discovery of shared slots

The middle and bottom sections of Table 3 show the results of the IAC algorithm with a threshold 0.22 on cosine similarity. We chose this threshold such that the number of labels in the resulting clusters is the closest to that chosen by the human annotator. We see that the proposed algorithm was able to discover most of the shared labels identified by the human annotator, with only *location* missing. Moreover, the algorithm additionally discovered *description* and *num-people*, both of which are reasonable clusters.

The best way to evaluate the proposed heuristics, however, is to inspect its impact on slot filling performance. We thus used the automatically discovered shared slots in our MTL algorithms. We only evaluated the JM approach since it gave the best overall performance in our previous experiments. We obtained 0.880, 0.877 and 0.788 in F1 for the three tasks respectively. The increase in F1 averaged over tasks is statistically significant than the baseline. This suggests that in practice we may be able to apply our multi-task learning strategies without human supervision.

6. Conclusions

This work is concerned with the problem of learning multiple slot filling tasks with different, yet overlapping, sets of slots. Our contribution in this work is two-folds. First, we assumed the knowledge of how the slots can be shared across tasks, and proposed four MTL algorithms that transfer such information across different tasks. These algorithms were evaluated and compared on 3 related slot filling tasks, where the JM and AD approaches significantly improved over single-task learning baselines and where the latter approach is more efficient computationally. Secondly, we presented an intra-domain clustering algorithm to discover shared slots from training data. Experiments on the same dataset showed the potential of an MTL system without human supervision.

7. References

[1] http://www.siri.com, *Siri, your virtual personal assistant*.

[2] http://itunes.apple.com/us/app/bing/id345323231?mt=8, *Bing for iPhone*.

[3] Rich Caruana, "Multitask learning," *Machine Learning Journal*, vol. 28, 1997.

[4] Gokhan Tur, "Multi-task learning for spoken language understanding," in *Proc. of ICASSP*, 2006.

[5] Charles Sutton and Andrew McCallum, "Composition of conditional random fields for transfer learning," in *Proceedings of HLT/EMNLP*, 2005.

[6] Sunita Sarawagi and William W. Cohen, "Semi-Markov conditional random fields for information extraction," in *NIPS*, 2004.

[7] Sameer Singh, Dustin Hillard, and Chris Leggetter, "Minimally-supervised extraction of entities from text advertisements," in *HLT*, 2010.

[8] Xiao Li, "Understanding the semantic structure of noun phrase queries," in *Proc. of ACL*, 2010.

[9] Ye-Yi Wang, "Strategies for statistical spoken language understanding with small amount of data - an empirical study," in *Interspeech*, 2010.

[10] Jingjing Liu, Xiao Li, Alex Acero, and Ye-Yi Wang, "Lexicon modeling for query understanding," in *Proc. of ICASSP*, 2011.

[11] Xiao Li, Ye-Yi Wang, and Alex Acero, "Extracting structured information from user queries with semi-supervised conditional random fields," in *Proc. of SIGIR'09*, 2009.