International Conference on Spoken Language Processing, pp. 1766-1769, ISCA, Pittsburgh, PA, USA, 2006

Discriminative Models for Spoken Language Understanding

Ye-Yi Wang and Alex Acero

Microsoft Research, Redmond, Washington, USA

{yeyiwang,alexac}@microsoft.com

Abstract

This paper studies several discriminative models for spoken language understanding (SLU). While all of them fall into the conditional model framework, different optimization criteria lead to conditional random fields, perceptron, minimum classification error and large margin models. The paper discusses the relationship amongst these models and compares them in terms of accuracy, training speed and robustness.

Index Terms: discriminative training, conditional random fields (CRFs), large margin (LM) training, MCE, perceptron, spoken language understanding (SLU).

1. Introduction

SLU addresses the problem of extracting semantic meaning conveyed in a user's utterance in a specific domain. A domain is often defined with frame-like structures, as the simplified example shown in Figure 1 for the Air Travel Information System (ATIS) domain [1]. The task of SLU is to map a user's utterance into a semantic representation, which is an instantiation of the semantic frames. An example of semantic representation is shown in Figure 2 for the utterance "Show me the flights departing from Seattle arriving at Washington D.C." or "Flights from Seattle to Washington D.C. please."

```
< frame name="ShowFlight" >
    < slot name="DCity" filler="City"/ >
    < slot name="ACity" filler="City"/ >
< /frame >
< frame name="GroundTrans" >
    < slot name="City" filler="City"/ >
< /frame >
```

Figure 1. *Simplified semantic frames in the ATIS domain. A frame represents a command (the name of the frame) in the domain and the slots associated with the command. The filler attribute of a slot specifies the semantic object that can fill a slot. For example, a string covered by the "City" rule in a context-free grammar (CFG) can be the filler of the ACity (ArriveCity) or the DCity (DepartCity) slot.*

```
< ShowFlight" >
    < DCity > Seattle < /DCity >
    < ACity > Washington D.C. < /ACity >
< /ShowFlight >
```

Figure 2. *Semantic representation is an instantiation of the semantic frames in Figure 1.*

SLU is traditionally solved with a knowledge-based approach. In the past decade many data-driven generative statistical models have been proposed for the problem [2]. Among them a HMM/CFG composite model [2] integrates knowledge-based approach in a statistical learning framework, which uses CFG rules to define the slot fillers and hidden Markov models (HMMs) to disambiguate command and slots in an utterance. The CFG rules may be obtained from a domain-independent library (e.g., for date and time expressions), or from a domain specific database (e.g., city and airport names.) The topology of the HMMs is determined by the domain defined by semantic frames. The inclusion of the prior knowledge in the statistical model compensates for the dearth of labeled data for model training. The HMM/CFG composite model achieves the understanding accuracy at the same level as the best performing semantic parsing system based on a manually developed grammar in ATIS evaluation [3].

The prior knowledge about a domain is similarly exploited in discriminative conditional models for further improving the understanding accuracy. We have shown that discriminative power and the capability of incorporating overlapping features in conditional random fields (CRFs) [4] has resulted in more than 20% slot error rate reduction for SLU over the generative HMM/CFG composite model [5].

We have recently investigated other discriminative models in the same log-linear framework to tackle the problem, including perceptron [6], minimum classification error (MCE) [7] and large margin (LM) [8]. This paper discusses the relationship amongst these models, compares their performance in terms of accuracy, training speed and robustness to data sparseness.

The paper is organized as follows. Section 2 lays down the conditional model framework for SLU. Section 3 introduces the different optimization criteria that result in CRFs, perceptron, MCE and LM models. Section 4 compares the models with experimental results, and Section 5 concludes the paper.

2. Conditional Models for SLU

We convert the SLU problem into a sequential labeling problem that assign a domain-related tag to each word in an utterance. For example, the semantic representation in Figure 2 can be recovered from the following label sequence:

"Flights/SF.DCity.pre from/SF.DCity.pre Seattle/SF.DCity.start to/SF.ACity.pre Washington/SF.ACity.start D.C./SF.ACity.cont please/SF.post"

Here the prefix "SF" in the labels indicates that the utterance is a "ShowFlight" command. "SF.DCity.pre" indicates that the word is a *preamble* for a DCity slot. "SF.ACity.start" represents the first word of an ACity slot filler, "SF.ACity.cont" represents continuation of an ongoing ACity slot, and "SF.post" is a *post-command* state that labels all the words after the last slot. Each label here corresponds to a state of the model, so "states" and "labels" will be used interchangeably. The model imposes a couple of constraints on state transitions, namely a preamble state must be followed by itself or by its corresponding slot's start state; and any state must be followed by a state in the same command (with the same prefix.)

Formally, the problem can be formulated as assigning a state sequence s_1^τ to an observation $\mathbf{o}$ with τ words with the help of a CFG parser that identifies all possible matches of CFG rules for slot fillers, as illustrated in Figure 3. In this example, the model needs to label "two" as the "SF.NumOfTickets.start" (slot not shown in the simplified frames in **Figure 1**) "Washington" as "SF.ACity.start," "D.C." as "SF.ACity.cont," and the remaining words as the appropriate preambles. To do so, the model has to resolve several types of ambiguities:

1. Filler/non-filler ambiguity, e.g., "two" can be the filler of a NumOfTickets slot, or the preamble of the ACity slot.
2. CFG ambiguity, e.g., "Washington" can be CFG-covered as either a City or a State.
3. Segmentation ambiguity, e.g., "[Washington] [D.C.]" for two Cities (or a State and a City) vs. "[Washington D.C.]" represents a single City.
4. Semantic label ambiguity, e.g., "Washington D.C." can fill either an "ACity" or a "DCity" slot.

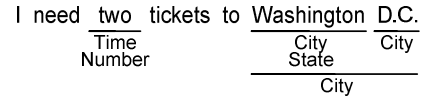

Figure 3. *A CFG parser identifies all possible matches of CFG rules for slot fillers.*

The desired state sequence s_1^τ should have maximum posterior probability $p(s_1^\tau \mid \mathbf{o}; \lambda)$ according to model λ. Here undirected conditional graphical models are used for the posterior. They are of the following form:

$$p(s_1^\tau \mid \mathbf{o}; \lambda) = \frac{1}{z(\mathbf{o}; \lambda)} \exp\left(\lambda \cdot \mathbf{f}(s_1^\tau, \mathbf{o})\right). \quad (1)$$

where $\mathbf{f}(s_1^\tau, \mathbf{o})$ is a vector of features that are functions of state sequence and observation; λ is a vector of parameters that are the weights for the features; and $z(\mathbf{o}; \lambda) = \sum_{s_1^\tau} \exp\left(\lambda \cdot \mathbf{f}(s_1^\tau, \mathbf{o})\right)$ normalizes the distribution over all possible state sequences. For computational tractability, it is often assumed that s_1^τ forms a Markov chain and each element feature f_k in $\mathbf{f}$ is a function that only depends on two adjacent states, so

$$p(s_1^\tau \mid \mathbf{o}; \lambda) = \frac{1}{z(\mathbf{o}; \lambda)} \exp\left(\sum_k \lambda_k \sum_{t=1}^{\tau} f_k(s^{(t-1)}, s^{(t)}, \mathbf{o}, t)\right) \quad (2)$$

The features we use for SLU in the conditional model include:

1. *Command prior* features capture the likelihood a command being issued by a user, e.g.,

$$f^{PR}_{\text{ShowFlight}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } t{=}0 \wedge \text{Prefix}(s^{(t)}) = \text{SF} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

2. *State Transition* features capture the ordering of different slots in a command, e.g.,

$$f^{TR}_{\text{SF.DCity,SF.ACity}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } s^{(t-1)} \in \{\text{SF.DCity.start, SF.DCity.cont}\} \\ & \quad \wedge\, s^{(t)} \in \{\text{SF.ACity.pre, SF.ACity.start}\} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

3. *Unigram* and *bigram* features capture the co-occurrence of words with preamble/post-command states, e.g.,

$$f^{UG}_{\text{SF.DCity.pre,from}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } s^{(t)}{=}\text{SF.DCity.pre} \wedge\ \mathbf{o}^t = \text{from}; \\ 0 & \text{otherwise} \end{cases}$$

$$f^{BG}_{\text{SF.DCity.pre,departing,from}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } s^{(t-1)}{=}s^{(t)}{=}\text{SF.DCity.pre} \\ & \quad \wedge\, \mathbf{o}^{t-1} = \text{departing} \wedge\ \mathbf{o}^t = \text{from} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

We include f^{BG}_{s,w_1w_2} in the model only for those bigrams w_1w_2 that co-occur with state s in a labeled training example.

4. *Previous slot's context* features with window size k capture the dependency of a slot's interpretation on the preamble of the preceding slot, e.g.,

$$f^{PC}_{\text{SF.ACity,SF.DTime},w}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } s^{(t-1)} \in \{\text{SF.ACity.start, SF.ACity.cont}\} \\ & \quad \wedge\, s^{(t)} \in \{\text{SF.DTime.start, SF.DTime.pre}\} \\ & \quad \wedge w \in \Theta(\text{SF.ACity}, \mathbf{o}, t-1, k) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Here $\Theta(\text{SF.ACity}, \mathbf{o}, t-1, k)$ represents a word set that contains up to k words, which are not covered by any CFG rules, and which appear in front of the longest phrases covered by the CFG rule for the slot SF.ACity. The utility of the features can be illustrated with utterances "Flights from Seattle to Boston at 3 pm" and "Flights from Seattle arriving in Boston at 3 pm." In the first utterance, "3 pm" is a DTime (Departure Time), while it is an ATime (Arrival Time) in the second one. The state transition features are not able to distinguish the two cases, since the state sequences before "3 pm" are exactly same. Only features like $f^{PC}_{\text{SF.ACity,SF.DTime,arriving}}$ or $f^{PC}_{\text{SF.ACity,SF.DTime,to}}$ can help distinguish the two different cases.

5. CFG *chunk coverage* features for preamble/post-command words capture the likelihood that a word covered by a CFG rule may not be part of a slot filler, e.g.,

$$f^{CC1}_{\text{ShowFlight,City}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{if } \text{Prefix}(s^{(t)}) = \text{SF} \\ & \quad \wedge \text{covers}(\text{City}, \mathbf{o}^t) \wedge\ \neg\text{IsFiller}(s^{(t)}) \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

Here IsFiller(s) indicates state s is a slot filler state – either a start state or a continuation state. The feature is activated when non-filler labeled $\mathbf{o}^t$ is covered by the City rule.

6. CFG *chunk coverage* features for slot boundaries prevent errors like segmenting "Washington D.C." into two different slots. They are activated when a slot boundary is covered by a CFG rule, e.g.,

$$f^{CC2}_{\text{ShowFlight,City}}(s^{(t-1)}, s^{(t)}, \mathbf{o}, t) = \begin{cases} 1 & \text{Prefix}(s^{(t)}) = \text{SF} \wedge \text{covers}(\text{City}, \mathbf{o}^t_{t-1}) \\ & \quad \wedge\ \text{isFiller}(s^{(t-1)}) \wedge\ \text{isFillerStart}(s^{(t)}) \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

3. Discriminative Training

Each feature f_k in the previous section is associated with a weight λ_k in Eq. (2). The parameters of the model can be optimized according to different objective functions, which yield the following discriminative training methods.

3.1. Conditional Random Fields (CRFs)

CRFs [4] maximize log posterior probability of training set label sequences given the observation:

$$\begin{aligned} L(\lambda) &= \sum_i \log P(\mathbf{s}_i \mid \mathbf{o}_i;\lambda) \\ &= \sum_i \sum_{f_k} \lambda_k \sum_{t=1}^{\|\mathbf{o}_i\|} f_k\left(s_i^{(t-1)}, s_i^{(t)}, \mathbf{o}_i, t\right) - \log Z_\lambda(\mathbf{o}_i) \end{aligned} \tag{9}$$

Here $\mathbf{o}_i, \mathbf{s}_i$ are the observation and the label sequence of the *i*-th training example. Eq. (9) can be optimized with gradient-based optimization like stochastic gradient decent . Its gradient is

$$\begin{aligned} \frac{\partial L(\lambda)}{\partial \lambda_k} &= \sum_i \sum_{t=1}^{\|\mathbf{o}_i\|} f_k\left(s_i^{(t-1)}, s_i^{(t)}, \mathbf{o}_i, t\right) \\ &\quad - \sum_i \mathbf{E}_{P(\mathbf{l}\mid\mathbf{o}_i)}\left[f_k\left(l^{(t-1)}, l^{(t)}, \mathbf{o}_i, t\right)\right] \end{aligned} \tag{10}$$

which is the difference between the counts of feature f_k from the labeled state-sequence/observation pairs and its expected count over all possible state-sequences given the observations alone. The expected count can be obtained with a forward-backward like dynamic programming algorithm.

Because $L(\lambda)$ is a convex function, a global optimum exists for the function. This property only holds for CRFs among the discriminative algorithms that we study in this paper.

3.2. Perceptron

Given training data pair $\mathbf{o}_i, \mathbf{s}_i$, if the Viterbi state sequence $\mathbf{v}_i$ for $\mathbf{o}_i$ is different from $\mathbf{s}_i$, perceptron learning [6] updates λ_k by adding the difference of the feature counts between label sequence $\mathbf{s}_i$ and $\mathbf{v}_i$: $\lambda_k' = \lambda_k + f_k(\mathbf{s}_i, \mathbf{o}_i, t) - f_k(\mathbf{v}_i, \mathbf{o}_i, t)$.

Perceptron learning can be viewed as using the difference of the log posterior probability between the reference state sequence and the Viterbi state sequence of mislabeled samples as the optimization objective function:

$$L'(\lambda) = \sum_{i:\mathbf{o}_i \neq \mathbf{v}_i} \log P(\mathbf{s}_i \mid \mathbf{o}_i;\lambda) - \max_{\mathbf{v}} \log P(\mathbf{v} \mid \mathbf{o}_i;\lambda) \tag{11}$$

The gradient of this function is the count difference used in perceptron parameter update. Therefore perceptron training is an online gradient-based optimization for Eq. (11). In this case, there is no need of dynamic programming for the expected feature counts. This speeds up model training significantly.

3.3. Minimum Classification Error (MCE)

MCE [7] directly minimizes the sequence label errors. Given training data $\mathbf{o}_i, \mathbf{s}_i$, a mislabeling measure is defined as

$$d(\lambda) = \max_{\mathbf{v}\neq\mathbf{s}_i} \log P(\mathbf{v} \mid \mathbf{o}_i;\lambda) - \log P(\mathbf{s}_i \mid \mathbf{o}_i;\lambda) \tag{12}$$

which is the log posterior difference between the best incorrect state sequence and the correct reference state sequence. It is positive when the best incorrect sequence has higher posterior, negative otherwise. A loss function then maps the mislabeling measure to a 0-1 continuum. Often a sigmoid function is used:

$$L''(\lambda) = \frac{1}{1+\exp(-\alpha d(\lambda) + \delta)} \tag{13}$$

The gradient of the loss function is

$$\frac{\partial L''(\lambda)}{\partial \lambda_k} = \alpha\left(1+e^{-\alpha(d(\lambda)+\delta)}\right)^{-2} e^{-\alpha(d(\lambda)+\delta)} \frac{\partial d(\lambda)}{\partial \lambda_k} \tag{14}$$

Here $\alpha\left(1+e^{-\alpha(d(\lambda)+\delta)}\right)^{-2} e^{-\alpha(d(\lambda)+\delta)}$ is a scaling factor that reduces the influence of an example when its mislabeling measure is far away from δ (often set to 0) – in this case it is likely that the example is an outlier. This effectively makes the learning focus on the decision boundary, and requires a good initial parameterization – otherwise a large portion of training data may be treated as outliers. We pre-train the model with CRFs or perceptron before MCE training. In Eq. (14), other than the scaling factor, $\partial d(\lambda)/\partial \lambda_k$ is very similar to the negative gradient of $L'(\lambda)$ in perceptron learning, except that perceptron does not learn from correctly labeled data (corrective learning in ASR) while MCE keeps pulling probability mass from the posterior of the best incorrect state sequence to that of the correct sequence. The negation is due to the fact that the objective function needs to be maximized for perceptron but minimized for MCE.

3.4. Large Margin (LM)

The margin around the decision boundary for a sample $\mathbf{o}_i, \mathbf{s}_i$ is the log posterior difference between the correct state sequence and the best incorrect state sequence:

$$m(\mathbf{s}_i, \mathbf{o}_i) = \log P(\mathbf{s}_i \mid \mathbf{o}_i;\lambda) - \max_{\mathbf{v}\neq\mathbf{s}_i} \log P(\mathbf{v} \mid \mathbf{o}_i;\lambda) \tag{15}$$

The objective function in LM training is the minimum margin across all training samples [8]:

$$L'''(\lambda) = \min_{i:m(\mathbf{s}_i,\mathbf{o}_i)>0} m(\mathbf{s}_i,\mathbf{o}_i) \tag{16}$$

Rather than fitting the model to training data, LM training draws a decision boundary that has the largest minimum margin to the training examples. That makes the model more robust when the training data is limited.

In Eq. (16), examples with negative margin (i.e., examples that are mislabeled by the model) are treated as outliers and do not contribute to model optimization. This constraint can be relaxed with the introduction of a slate variable. In this paper we simply discard the examples with negative margins – this then requires that the initial model makes as few mistakes on the training data as possible. For that reason, we pre-train the model with CRFs or perceptron training before LM training.

To speedup training, LM learns from examples with margins smaller than a threshold instead of only learning from the example with the minimum positive margin.

4. Experimental Results

The ATIS 3 category A training set (~1700 utterances), as well the 1993 test (470 utterances, with 1702 slots) and development (410 utterances) sets are manually annotated for the experiment. Stochastic Gradient Decent (SGD) [9] is used for optimization. The development set are used to tune the training parameters and for stopping criteria for model training. The annotation

contains sufficient semantic information to achieve near perfect database query results when SQL queries are generated from the annotation. The only few "errors" of query results are due to either reference mistakes or the misplacement of category D test utterances (interpretation depends on discourse) in category A.

4.1. SLU Accuracy

Table 1 compares the SLU slot error rate with different feature sets across different models. All feature sets include 6 command prior, 1377 state transition, and 14433 unigram features. FSet1 also includes 290 previous slot's context feature with window size 1, and 156 chunk coverage features for preamble/post-command words. FSet2 includes additional chunk coverage features for slot boundaries, which share the weights with the chunk coverage features for preamble/post-command words. FSet3 adds on top of FSet2 58191 bigram features and uses previous slot's context feature with window size 2. A detailed study of the impacts of different features can be found in [5]. Both MCE and LM training are initialized in two ways – 40 iterations of perceptron training and 250 iterations of CRF training. At these iterations the initialization training methods accomplish close-to-optimal accuracy on the development set. The differences of slot error rates are not significant across different training methods except for perceptron learning, which has significantly higher error rates with FSet1 and FSet3, and perceptron-initialized MCE on FSet1. All the discriminative models significantly reduce the 5.0% slot error rate achieved by the generative HMM/CFG composite model.

While MCE and LM training improve the initial model trained with perceptron, they make little improvement over the initial model trained with CRFs.

Table 1. *SLU slot ins-del-sub error rates.*

Model	FSet 1	FSet 2	FSet 3
CRFs	4.00%	4.11%	3.88%
Perceptron	4.35%	3.94%	4.47%
Ptron/MCE	4.35%	3.94%	4.17%
CRF/MCE	3.94%	4.11%	3.88%
Ptron./LM	4.17%	4.11%	4.11%
CRFs/LM	4.05%	4.11%	3.88%

4.2. Training Speed

Table 2 compares the training speeds of different algorithms. Perceptron takes much fewer iterations and much less time per iteration than CRFs. Since initialization takes most of the training time for MCE and LM, perceptron initialized MCE and LM training are also much faster than CRF-initialized training. MCE converges faster than LM after perceptron initialization. This may due to the fact that all data participate in MCE training, while only those with small positive margins participate in LM training.

Table 2. *FSet2 training time per iteration, number of iterations for convergence and total training time.*

Model	Time/Iter.	Iteration	Total Time
CRFs	4.8s	320	26 min.
Perceptron	1.5s	40	1 min.
Ptron/MCE	1.9s	40+10	1.6 min.
CRFs/MCE	4.6s	250+30	22 min.
Ptron/LM	1.8s	40+60	3 min.
CRFs/LM	4.2s	250+30	20 min.

4.3. Robustness to Data Sparseness

Table 3 compares the accuracy of different models when only part of the data is used for model training. CRFs and CRF-initialized MCE/LM models are more robust to data sparseness than perceptron and perceptron-initialized MCE/LM models.

Table 3. *FSet2 Slot error rates of models trained with different amount of data.*

Model	¼ Data	½ Data	¾ Data	All Data
CRFs	5.64%	4.23%	4.17%	4.11%
Perceptron	7.17%	5.82%	5.23%	3.94%
Ptron/MCE	6.05%	5.52%	5.52%	3.94%
CRFs/MCE	5.88%	4.41%	4.17%	4.11%
Ptron/LM	6.64%	5.52%	3.82%	4.11%
CRFs/LM	5.93%	4.41%	4.17%	4.11%

5. Conclusions

We have introduced different discriminative training criteria for conditional models for SLU, and compared CRFs, perceptron, MCE and LM models in terms of accuracy, training speed and robustness to data sparseness. All the discriminative models accomplished similar accuracy except for perceptron, which has significantly higher error rates. CRFs are also more robust when less training data is available. However, perceptron and perceptron-initialized MCE and LM models are much faster to train, and the accuracy gap becomes smaller when more data are available. It is a good tradeoff to use perceptron-initialized MCE or LM if training speed is crucial for an application, or a large amount of training data is available.

6. References

[1] Price, P. Evaluation of Spoken Language System: the ATIS domain. DARPA Speech and Natural Language Workshop. 1990. Hidden Valley, PA.

[2] Wang, Y.-Y., L. Deng, and A. Acero, Spoken Language Understanding. IEEE Signal Processing Magazine, 2005. 22(5): p. 16-31.

[3] Ward, W. Recent Improvements in the CMU Spoken Language Understanding System. Human Language Technology Workshop. 1994. Plainsboro, NJ.

[4] Lafferty, J., A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. ICML 2001.

[5] Wang, Y.-Y., et al. Combining Statistical and Knowledge-based Spoken Language Understanding in Conditional Models. COLING/ACL 2006.

[6] Collins, M. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. EMNLP 2002.

[7] Juang, B.-H. and S. Katagiri, Discriminative Learning for Minimum Error Classification. IEEE Transaction on Signal Processing, 1992. 40(12): p. 3043-3054.

[8] Li, X., H. Jiang, and C. Liu. Large Margin HMMs for Speech Recognition. ICASSP. 2005. Philadelphia, PA.

[9] Kushner, H.J. and G.G. Yin, Stochastic Approximation Algorithms and Applications. 1997: Springer-Verlag.