In Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL-2010), pp. 203--212, Uppsala, Sweden, July 2010.

Joint Entity and Relation Extraction using Card-Pyramid Parsing

Rohit J. Kate and Raymond J. Mooney
Department of Computer Science
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
{rjkate,mooney}@cs.utexas.edu

Abstract

Both entity and relation extraction can benefit from being performed jointly, allowing each task to correct the errors of the other. We present a new method for joint entity and relation extraction using a graph we call a "card-pyramid." This graph compactly encodes all possible entities and relations in a sentence, reducing the task of their joint extraction to jointly labeling its nodes. We give an efficient labeling algorithm that is analogous to parsing using dynamic programming. Experimental results show improved results for our joint extraction method compared to a pipelined approach.

1 Introduction

Information extraction (IE) is the task of extracting structured information from text. The two most common sub-tasks of IE are extracting entities (like *Person*, *Location* and *Organization*) and extracting relations between them (like *Work_For* which relates a *Person* and an *Organization*, *OrgBased_In* which relates an *Organization* and a *Location* etc.). Figure 1 shows a sample sentence annotated with entities and relations. The application domain and requirements of the downstream tasks usually dictate the type of entities and relations that an IE system needs to extract.

Most work in IE has concentrated on entity extraction alone (Tjong Kim Sang, 2002; Sang and Meulder, 2003) or on relation extraction assuming entities are either given or previously extracted (Bunescu et al., 2005; Zhang et al., 2006; Giuliano et al., 2007; Qian et al., 2008). However, these tasks are very closely inter-related. While identifying correct entities is essential for identifying relations between them, identifying correct relations can in turn improve identification of entities. For example, if the relation *Work_For* is identified with high confidence by a relation extractor, then it can enforce identifying its arguments as *Person* and *Organization*, about which the entity extractor might not have been confident.

A brute force algorithm for finding the most probable joint extraction will soon become intractable as the number of entities in a sentence grows. If there are n entities in a sentence, then there are $O(n^2)$ possible relations between them and if each relation can take l labels then there are $O(l^{n^2})$ total possibilities, which is intractable even for small l and n. Hence, an efficient inference mechanism is needed for joint entity and relation extraction.

The only work we are aware of for jointly extracting entities and relations is by Roth & Yih (2004; 2007). Their method first identifies the possible entities and relations in a sentence using separate classifiers which are applied independently and then computes a most probable consistent global set of entities and relations using linear programming. In this paper, we present a different approach to joint extraction using a "card-pyramid" graph. The labeled nodes in this graph compactly encode the possible entities and relations in a sentence. The task of joint extraction then reduces to finding the most probable joint assignment to the nodes in the card-pyramid. We give an efficient dynamic-programming algorithm for this task which resembles CYK parsing for context-free grammars (Jurafsky and Martin, 2008). The algorithm does a beam search and gives an approximate solution for a finite beam size. A natural advantage of this approach is that extraction from a part of the sentence is influenced by extraction from its subparts and vice-versa, thus leading to a joint extraction. During extraction from a part of the sentence it also allows use of features based on the extraction from its sub-parts, thus leading to a more integrated extraction. We use Roth & Yih's

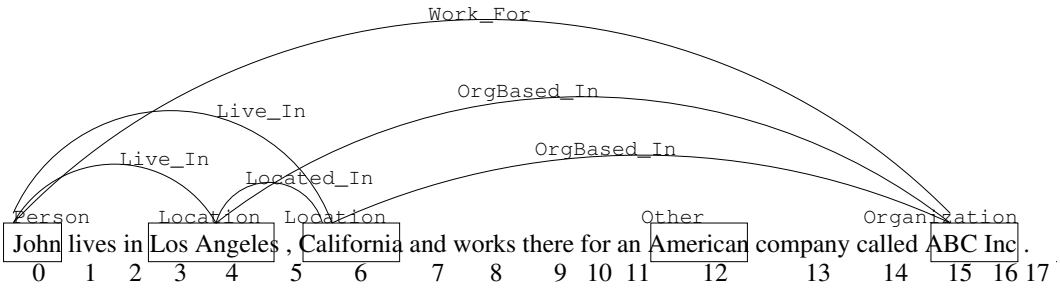

Figure 1: A sentence shown with entities and relations.

Figure 2: A pyramid built out of playing-cards.

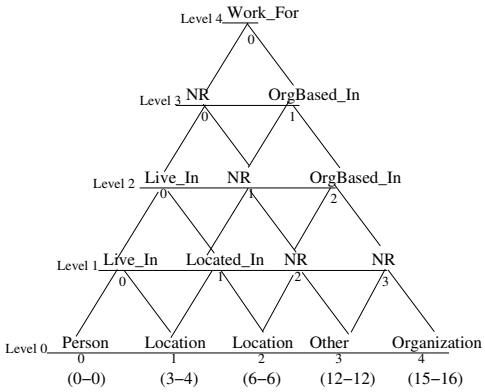

Figure 3: The card-pyramid for the sentence shown in Figure 1. Levels are shown by the horizontal lines under which the positions of its nodes are indicated.

(2004; 2007) dataset in our experiments and show that card-pyramid parsing improves accuracy over both their approach and a pipelined extractor.

2 Card-Pyramid Parsing for Joint Extraction

In this section, we first introduce the card-pyramid structure and describe how it represents entities and their relations in a sentence. We then describe an efficient algorithm for doing joint extraction using this structure.

2.1 Card-Pyramid Structure

We define a binary directed graph we call a *card-pyramid* because it looks like a pyramid built out of playing-cards as shown in Figure 2. A card-pyramid is a "tree-like" graph with one root, internal nodes, and leaves, such that if there are n leaves, then there are exactly n levels with a decreasing number of nodes from bottom to top, leaves are at the lowest level (0) and the root is at the highest level $(n-1)$ (see Figure 3 for an example). In addition, every non-leaf node at position i in level l is the parent of exactly two nodes at positions i and $i+1$ at level $l-1$. Note that a card-pyramid is not a tree because many of its nodes have two parents. A useful property of a card-pyramid is that a non-leaf node at position i in level l is always the lowest common ancestor of the leaves at positions i and $l+i$.

We now describe how entities and relations in a sentence are easily represented in a card-pyramid. We assume that in addition to the given entity types, there is an extra type, *Other*, indicating that the entity is of none of the given types. Similarly, there is an extra relation type, *NR*, indicating that its two entity arguments are not related.

Figure 3 shows the card-pyramid corresponding to the annotated sentence shown in figure 1. To obtain it, first, all entities present in the sentence are made leaves of the card-pyramid in the same order as they appear in the sentence. The label of a leaf is the type of the corresponding entity. The leaf also stores the range of the indices of its entity's words in the sentence. Note that although there is no overlap between entities in the given example (nor in the dataset we used for our experiments),

overlapping entities do not pose a problem. Overlapping entities can still be ordered and supplied as the leaves of the card-pyramid. Next, the relation between every two entities (leaves) is encoded as the label of their lowest common ancestor. If two entities are not related, then the label of their lowest common ancestor is *NR*. This way, every non-leaf node relates exactly two entities: the left-most and right-most leaves beneath it.

2.2 Card-Pyramid Parsing

The task of jointly extracting entities and relations from a sentence reduces to jointly labeling the nodes of a card-pyramid which has all the candidate entities (i.e. entity boundaries) of the sentence as its leaves. We call this process card-pyramid parsing. We assume that all candidate entities are given up-front. If needed, candidate entities can be either obtained automatically (Punyakanok and Roth, 2001) or generated using a simple heuristic, like including all noun-phrase chunks as candidate entities. Or, in the worst case, each substring of words in the sentence can be given as a candidate entity. Liberally including candidate entities is possible since they can simply be given the label *Other* if they are none of the given types.

In this section we describe our card-pyramid parsing algorithm whose pseudo-code is shown in Figure 4. While the process is analogous to context-free grammar (CFG) parsing, particularly CYK bottom-up parsing, there are several major differences. Firstly, in card-pyramid parsing the structure is already known and the only task is labeling the nodes, whereas in CFG parsing the structure is not known in advance. This fact simplifies some aspects of card-pyramid parsing. Secondly, in CFG parsing the subtrees under a node do not overlap which simplifies parsing. However, in card-pyramid parsing there is significant overlap between the two sub-card-pyramids under a given node and this overlap needs to be consistently labeled. This could have potentially complicated parsing, but there turns out to be a simple constant-time method for checking consistency of the overlap. Thirdly, while CFG parsing parses the words in a sentence, here we are parsing candidate entities. Finally, as described below, in card-pyramid parsing, a production at a non-leaf node relates the left-most and right-most leaves beneath it, while in CFG parsing a production at a non-leaf node relates its immediate children which could be other non-leaf nodes.

Parsing requires specifying a grammar for the card-pyramid. The productions in this grammar are of two types. For leaf nodes, the productions are of the form *entityLabel* $\rightarrow$ *ce* where *ce*, which stands for candidate entity, is the only terminal symbol in the grammar. We call these productions *entity productions*. For non-leaf nodes, the productions are of the form *relationLabel* $\rightarrow$ *entityLabel1 entityLabel2*. We call these productions *relation productions*. Note that their right-hand-side (RHS) non-terminals are entity labels and not other relation labels. From a training set of labeled sentences, the corresponding card-pyramids can be constructed using the procedure described in the previous section. From these card-pyramids, the entity productions are obtained by simply reading off the labels of the leaves. A relation productions is obtained from each non-leaf node by making the node's label the production's left-hand-side (LHS) non-terminal and making the labels of its left-most and right-most leaves the production's RHS non-terminals. For the example shown in Figure 3, some of the productions are *Work_For* $\rightarrow$ *Person Organization*, *NR* $\rightarrow$ *Person Other*, *OrgBased_In* $\rightarrow$ *Loc Org*, *Person* $\rightarrow$ *ce*, *Location* $\rightarrow$ *ce* etc. Note that there could be two separate productions like *Work_For* $\rightarrow$ *Person Organization* and *Work_For* $\rightarrow$ *Organization Person* based on the order in which the entities are found in a sentence. For the relations which take arguments of the same type, like *Kill(Person,Person)*, two productions are used with different LHS non-terminals (*Kill* and *Kill_reverse*) to distinguish between the argument order of the entities.

The parsing algorithm needs a classifier for every entity production which gives the probability of a candidate entity being of the type given in the production's LHS. In the pseudo-code, this classifier is given by the function: *entity-classifier(production, sentence, range)*. The function *range(r)* represents the boundaries or the range of the word indices for the rth candidate entity. Similarly, we assume that a classifier is given for every relation production which gives the probability that its two RHS entities are related by its LHS relation. In the pseudo-code it is the function: *relation-classifier(production, sentence, range1, range2, sub-card-pyramid1, sub-card-pyramid2)*, where *range1* and *range2* are the

ranges of the word indices of the two entities and *sub-card-pyramid1* and *sub-card-pyramid2* are the sub-card-pyramids rooted at its two children. Thus, along with the two entities and the words in the sentence, information from these sub-card-pyramids is also used in deciding the relation at a node. In the next section, we further specify these entity and relation classifiers and explain how they are trained. We note that this use of multiple classifiers to determine the most probable parse is similar to the method used in the KRISP semantic parser (Kate and Mooney, 2006).

Given the candidate entities in a sentence, the grammar, and the entity and relation classifiers, the card-pyramid parsing algorithm tries to find the most probable joint-labeling of all of its nodes, and thus jointly extracts entities and their relations. The parsing algorithm does a beam search and maintains a *beam* at each node of the card-pyramid. A node is represented by $l[i][j]$ in the pseudo-code which stands for the node in the jth position in the ith level. Note that at level i, the nodes range from $l[i][0]$ to $l[i][n-i-1]$, where n is the number of leaves. The beam at each node is a queue of items we call *beam elements*. At leaf nodes, a beam element simply stores a possible entity label with its corresponding probability. At non-leaf nodes, a beam element contains a possible joint assignment of labels to all the nodes in the sub-card-pyramid rooted at that node with its probability. This is efficiently maintained through indices to the beam elements of its children nodes.

The parsing proceeds as follows. First, the entity classifiers are used to fill the beams at the leaf nodes. The *add(beam, beam-element)* function adds the beam element to the beam while maintaining its maximum beam-width size and sorted order based on the probabilities. Next, the beams of the non-leaf nodes are filled in a bottom-up manner. At any node, the beams of its children nodes are considered and every combination of their beam elements are tried. To be considered further, the two possible sub-card-pyramids encoded by the two beam elements must have a consistent overlap. This is easily enforced by checking that its left child's right child's beam element is same as its right child's left child's beam element. If this condition is satisfied, then those relation productions are considered which have the left-most leaf of the left child and right-most leaf of the right child as its RHS non-terminals.[1] For every such production in the grammar, [2] the probability of the relation is determined using the relation classifier. This probability is then multiplied by the probabilities of the children sub-card-pyramids. But, because of the overlap between the two children, a probability mass gets multiplied twice. Hence the probability of the overlap sub-card-pyramid is then suitably divided. Finally, the estimated most-probable labeling is obtained from the top beam element of the root node.

We note that this algorithm may not find the optimal solution but only an approximate solution owing to a limited beam size, this is unlike probabilistic CFG parsing algorithms in which the optimal solution is found. A limitless beam size will find the optimal solution but will reduce the algorithm to a computationally intractable brute force search. The parsing algorithm with a finite beam size keeps the search computationally tractable while allowing a joint labelling.

3 Classifiers for Entity and Relation Extraction

The card-pyramid parsing described in the previous section requires classifiers for each of the entity and relation productions. In this section, we describe the classifiers we used in our experiments and how they were trained.

We use a support vector machine (SVM) (Cristianini and Shawe-Taylor, 2000) classifier for each of the entity productions in the grammar. An entity classifier gets as input a sentence and a candidate entity indicated by the range of the indices of its words. It outputs the probability that the candidate entity is of the respective entity type. Probabilities for the SVM outputs are computed using the method by Platt (1999). We use all possible word subsequences of the candidate entity words as implicit features using a word-subsequence kernel (Lodhi et al., 2002). In addition, we use the following standard entity extraction features: the part-of-speech (POS) tag sequence of the candidate entity words, two words before and after the candidate entity and their POS tags, whether any or all candidate entity words are capitalized,

[1] These are stored in the beam elements.

[2] Note that this step enforces the consistency constraint of Roth and Yih (Roth and Yih, 2004; Roth and Yih, 2007) that a relation can only be between the entities of specific types. The grammar in our approach inherently enforces this constraint.

function Card-Pyramid-Parsing(Sentence,Grammar,entity-classifiers,relation-classifiers)
n = number of candidate entities in S
// Let range(r) represent the range of the indices of the words for the rth candidate entity.
// Let $l[i][j]$ represent the jth node at ith level in the card-pyramid.

// For leaves
// A beam element at a leaf node is (label,probability).
for $j = 0$ to n // for every leaf
  for each $entityLabel \rightarrow candidate\_entity \in$ Grammar
    prob = entity-classifier($entityLabel \rightarrow candidate\_entity$, S, range($j$))
    add($l[0][j].beam$, ($entityLabel$,prob))

// For non-leaf nodes
// A beam element at a non-leaf node is (label,probability,leftIndex,rightIndex,leftMostLeaf,rightMostLeaf)
// where leftIndex and rightIndex are the indices in the beams of the left and right children respectively.
for $i = 1$ to n // for every level above the leaves
  for $j = 0$ to $n - i - 1$ // for every position at a level
    // for each combination of beam elements of the two children
    for each $f \in l[i-1][j].beam$ and $g \in l[i-1][j+1].beam$
      // the overlapped part must be same (overlap happens for $i > 1$)
      if ($i == 1 || f.rightIndex == g.leftIndex$)
        for each $relationLabel \rightarrow f.leftMostLeaf\ g.rightMostLeaf \in$ Grammar
          // probability of that relation between the left-most and right-most leaf under the node
          prob = relation-classifier($relationLabel \rightarrow f.leftMostLeaf\ g.rightMostLeaf$, S, range($i$), range($i+j$), f, g);
          prob *= $f.probability * g.probability$ // multiply probabilities of the children sub-card-pyramids
          // divide by the common probability that got multiplied twice
          if ($i > 1$) prob /= $l[i-2][j+1].beam[f.rightIndex].probability$
          add($l[i][j].beam$, ($relationLabel$, prob, index of f, index of g, $f.leftMostLeaf$, $g.rightMostLeaf$)

return the labels starting from the first beam element of the root i.e. $l[n][0].beam[0]$

Figure 4: Card-Pyramid Parsing Algorithm.

whether any or all words are found in a list of entity names, whether any word has sufffix "ment" or "ing", and finally the alphanumeric pattern of characters (Collins, 2002) of the last candidate entity word obtained by replacing each character by its character type (lowercase, uppercase or numeric) and collapsing any consecutive repetition (for example, the alphanumeric pattern for CoNLL2010 will be AaA0). The full kernel is computed by adding the word-subsequence kernel and the dot-product of all these features, exploiting the convolution property of kernels.

We also use an SVM classifier for each of the relation productions in the grammar which outputs the probability that the relation holds between the two entities. A relation classifier is applied at an internal node of a card-pyramid. It takes the input in two parts. The first part is the sentence and the range of the word indices of its two entities l and r which are the left-most and the right-most leaves under it respectively. The second part consists of the sub-card-pyramids rooted at the node's two children which represent a possible entity and relation labeling for all the nodes underneath. In general, any information from the sub-card-pyramids could be used in the classifier. We use the following information: pairs of relations that exist between l and b and between b and r for every entity (leaf) b that exists between the two entities l and r. For example, in figure 3, the relation classifier at the root node which relates *Person(0-0)* and *Organization (15-16)* will take three pairs of relations as the information from the two sub-card-pyramids of its children: "*Live_In—OrgBased_In*" (with *Location(3-4)* as the in-between entity), "*Live_In—OrgBased_In*" (with *Location(6-6)* as the in-between entity) and "*NR—NR*" (with *Other(12-12)* as the in-between entity). This information tells how the two entities are related to the entities present in between them. This can affect the relation between the two entities, for example, if the sentence mentions that a person lives at a location and also mentions that an organization is based at that location then that person is likely to work at that organization. Note that this information can not be incorporated in a pipelined approach in which each relation is determined independently. It is also not possible to incorporate this in the linear programming method presented in (Roth and Yih, 2004; Roth and Yih, 2007) because that method computes the probabilities of all the relations independently before finding the optimal solution through linear programming. It would also not help to add hard constraints to their linear program relating the relations because they need not always hold.

We add the kernels for each part of the input to compute the final kernel for the SVM classifiers. The kernel for the second part of the input is computed by simply counting the number of common

pairs of relations between two examples thus implicitly considering every pair of relation (as described in the last paragraph) as a feature. For the first part of the input, we use word-subsequence kernels which have shown to be effective for relation extraction (Bunescu and Mooney, 2005b). We compute the kernel as the sum of the word-subsequence kernels between: the words between the two entities (*between pattern*), k (a parameter) words before the first entity (*before pattern*), k words after the second entity (*after pattern*) and the words from the beginning of the first entity to the end of the second entity (*between-and-entity pattern*). The *before*, *between* and *after* patterns have been found useful in previous work (Bunescu and Mooney, 2005b; Giuliano et al., 2007). Sometimes the words of the entities can indicate the relations they are in, hence we also use the *between-and-entity* pattern. When a relation classifier is used at a node, the labels of the leaves beneath it are already known, so we replace candidate entity words that are in the *between* and *between-and-entity*[3] patterns by their entity labels. This provides useful information to the relation classifier and also makes these patterns less sparse for training.

Given training data of sentences annotated with entities and relations, the positive and negative examples for training the entity and relation classifiers are collected in the following way. First, the corresponding card-pyramids are obtained for each of the training sentences as described in section 2.1. For every entity production in a card-pyramid, a positive example is collected for its corresponding classifier as the sentence and the range of the entity's word indices. Similarly, for every relation production in a card-pyramid, a positive example is collected for its corresponding classifier as the sentence, the ranges of the two entities' word indices and the sub-card-pyramids rooted at its two children. The positive examples of a production become the negative examples for all those productions which have the same right-hand-sides but different left-hand-sides. We found that for NR productions, training separate classifiers is harmful because it has the unwanted side-effect of preferring one label assignment of entities over another due to the fact that these productions gave different probabilities for the "not-related" relation between the entities. To avoid this, we found that it suffices if all these classifiers for NR productions always return 0.5 as the probability. This ensures that a real relation will be preferred over NR if and only if its probability is greater than 0.5, otherwise nothing will change.

[3]Except for the two entities at the ends

4 Experiments

We conducted experiments to compare our card-pyramid parsing approach for joint entity and relation extraction to a pipelined approach.

4.1 Methodology

We used the dataset[4] created by Roth & Yih (2004; 2007) that was also used by Giuliano et el. (2007). The sentences in this data were taken from the TREC corpus and annotated with entities and relations. As in the previous work with this dataset, in order to observe the interaction between entities and relations, our experiments used only the 1437 sentences that include at least one relation. The boundaries of the entities are already supplied by this dataset. There are three types of entities: *Person* (1685), *Location* (1968) and *Organization* (978), in addition there is a fourth type *Other* (705), which indicates that the candidate entity is none of the three types. There are five types of relations: *Located_In* (406) indicates that one *Location* is located inside another *Location*, *Work_For* (394) indicates that a *Person* works for an *Organization*, *OrgBased_In* (451) indicates that an *Organization* is based in a *Location*, *Live_In* (521) indicates that a *Person* lives at a *Location* and *Kill* (268) indicates that a *Person* killed another *Person*. There are 17007 pairs of entities that are not related by any of the five relations and hence have the *NR* relation between them which thus significantly outnumbers other relations.

Our implementation uses the LIBSVM[5] software for SVM classifiers. We kept the noise penalty parameter of SVM very high (100) assuming there is little noise in our data. For the word-subsequence kernel, we set 5 as the maximum length of a subsequence and 0.25 as the penalty parameter for subsequence gaps (Lodhi et al., 2002). We used $k = 5$ words for *before* and *after* patterns for the relation classifiers. These parameter values were determined through pilot experiments on a subset of the data. We used a beam

[4]Available at: http://l2r.cs.uiuc.edu/~cogcomp/Data/ER/conll04.corp

[5]http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Entity	*Person*			*Location*			*Organization*		
Approach	Rec	Pre	F	Rec	Pre	F	Rec	Pre	F
Pipeline	93.6	92.0	92.8	94.0	90.3	92.1	87.9	90.6	89.2
Card-pyramid	94.2	92.1	93.2	94.2	90.8	**92.4**†	88.7	90.5	89.5
RY07 Pipeline	89.1	88.7	88.6	88.1	89.8	88.9	71.4	89.3	78.7
RY07 Joint	89.5	89.1	**89.0**	88.7	89.7	**89.1**	72.0	89.5	79.2

Relation	*Located_In*			*Work_For*			*OrgBased_In*			*Live_In*			*Kill*		
Approach	Rec	Pre	F	Rec	Pre	F	Rec	Pre	F	Rec	Pre	F	Rec	Pre	F
Pipeline	57.0	71.5	62.3	66.0	74.1	69.7	60.2	70.6	64.6	56.6	68.1	61.7	61.2	91.1	73.1
Card-pyramid	56.7	67.5	58.3	68.3	73.5	70.7	64.1	66.2	64.7	60.1	66.4	**62.9**†	64.1	91.6	**75.2**
RY07 Pipeline	56.4	52.5	50.7	44.4	60.8	51.2	42.1	77.8	54.3	50.0	58.9	53.5	81.5	73.0	76.5
RY07 Joint	55.7	53.9	51.3	42.3	72.0	**53.1**	41.6	79.8	54.3	49.0	59.1	53.0	81.5	77.5	**79.0**†

Table 1: Results of five-fold cross-validation for entity and relation extraction using pipelined and joint extraction. Boldface indicates statistical significance ($p < 0.1$ using paired t-test) when compared to the corresponding value in the other row grouped with it. Symbol † indicates statistical significance with $p < 0.05$. Only statistical significance for F-measures are indicated. RY07 stands for the "E $\leftrightarrow$ R" model in (Roth and Yih, 2007).

size of 5 in our card-pyramid parsing algorithm at which the performance plateaus.

We note that by using a beam size of 1 and by not using the second part of input for relation classifiers as described in section 3 (i.e. by ignoring the relations at the lower levels), the card-parsing algorithm reduces to the traditional pipelined approach because then only the best entity label for each candidate entity is considered for relation extraction. Hence, in our experiments we simply use this setting as our pipelined approach.

We performed a 5-fold cross-validation to compare with the previous work with this dataset by Roth & Yih (2007), however, our folds are not same as their folds which were not available. We also note that our entity and relation classifiers are different from theirs. They experimented with several models to see the effect of joint inference on them, we compare with the results they obtained with their most sophisticated model which they denote by "E $\leftrightarrow$ R". For every entity type and relation type, we measured *Precision* (percentage of output labels correct), *Recall* (percentage of gold-standard labels correctly identified) and *F-measure* (the harmonic mean of *Precision* and *Recall*).

4.2 Results and Discussion

Table 1 shows the results of entity and relation extraction. The statistical significance is shown only for F-measures. We first note that except for the *Kill* relation, all the results of our pipelined approach are far better than the pipelined approach of (Roth and Yih, 2007), for both entities and relations. This shows that the entity and relation classifiers we used are better that the ones they used. These strong baselines also set a higher ceiling for our joint extraction method to improve upon.

The entity extraction results show that on all the entities the card-pyramid parsing approach for joint extraction obtains a better performance than the pipelined approach. This shows that entity extraction benefits when it is jointly done with relation extraction. Joint extraction using card-pyramid parsing also gave improvement in performance on all the relations except the *Located_In* relation.[6]

The results thus show that entity and relation extraction correct some of each other's errors when jointly performed. Roth & Yih (2004; 2007) report that 5% to 25% of the relation predictions of their pipeline models were *incoherent*, meaning that the types of the entities related by the relations are not of the required types. Their joint inference method corrects these mistakes, hence a part of the improvement their joint model obtains over their pipeline model is due to the fact that their pipeline model can output incoherent relations. Since the types of the entities a relation's arguments should

[6]Through error analysis we found that the drop in the performance for this relation was mainly because of an unusual sentence in the data which had twenty *Location* entities in it separated by commas. After incorrectly extracting *Located_In* relation between the *Location* entities at the lower levels, these erroneous extractions would be taken into account at higher levels in the card-pyramid, leading to extracting many more incorrect instances of this relation while doing joint extraction. Since this is the only such sentence in the data, when it is present in the test set during cross-validation, the joint method never gets a chance to learn not to make these mistakes. The drop occurs in only that one fold and hence the overall drop is not found as statistically significant despite being relatively large.

take are known, we believe that filtering out the incoherent relation predictions of their pipeline model can improve its precision without hurting the recall. On the other hand our pipelined approach never outputs incoherent relations because the grammar of relation productions enforce that the relations are always between entities of the required types. Thus the improvement obtained by our joint extraction method over our pipelined approach is always non-trivial.

5 Related Work

To our knowledge, Roth & Yih (2004; 2007) have done the only other work on joint entity and relation extraction. Their method employs independent entity and relation classifiers whose outputs are used to compute a most probable consistent global set of entities and relations using linear programming. One key advantage of our card-pyramid method over their method is that the classifiers can take the output of other classifiers under its node as input features during parsing. This is not possible in their approach because all classifier outputs are determined before they are passed to the linear program solver. Thus our approach is more integrated and allows greater interaction between dependent extraction decisions.

Miller et al. (2000) adapt a probabilistic context-free parser for information extraction by augmenting syntactic labels with entity and relation labels. They thus do a joint syntactic parsing and information extraction using a fixed template. However, as designed, such a CFG approach cannot handle the cases when an entity is involved in multiple relations and when the relations crisscross each other in the sentence, as in Figure 1. These cases occur frequently in the dataset we used in our experiments and many other relation-extraction tasks.

Giuliano et al. (2007) thoroughly evaluate the effect of entity extraction on relation extraction using the dataset used in our experiments. However, they employ a pipeline architecture and did not investigate joint relation and entity extraction. Carlson et al. (2009) present a method to simultaneously do semi-supervised training of entity and relation classifiers. However, their coupling method is meant to take advantage of the available unsupervised data and does not do joint inference.

Riedel et al. (2009) present an approach for extracting bio-molecular events and their arguments using Markov Logic. Such an approach could also be adapted for jointly extracting entities and their relations, however, this would restrict entity and relation extraction to the same machine learning method that is used with Markov Logic. For example, one would not be able to use kernel-based SVM for relation extraction, which has been very successful at this task, because Markov Logic does not support kernel-based machine learning. In contrast, our joint approach is independent of the individual machine learning methods for entity and relation extraction, and hence allows use of the best machine learning methods available for each of them.

6 Future Work

There are several possible directions for extending the current approach. The card-pyramid structure could be used to perform other language-processing tasks jointly with entity and relation extraction. For example, co-reference resolution between two entities within a sentence can be easily incorporated in card-pyramid parsing by introducing a production like $coref \rightarrow Person\ \ Person$, indicating that the two person entities are the same.

In this work, and in most previous work, relations are always considered between two entities. However, there could be relations between more than two entities. In that case, it should be possible to binarize those relations and then use card-pyramid parsing. If the relations are between relations instead of between entities, then card-pyramid parsing can handle it by considering the labels of the immediate children as RHS non-terminals instead of the labels of the left-most and the right-most leaves beneath it. Thus, it would be interesting to apply card-pyramid parsing to extract higher-order relations (such as causal or temporal relations).

Given the regular graph structure of the card-pyramid, it would be interesting to investigate whether it can be modeled using a probabilistic graphical model (Koller and Friedman, 2009). In that case, instead of using multiple probabilistic classifiers, one could employ a single jointly-trained probabilistic model, which is theoretically more appealing and might give better results.

Finally, we note that a better relation classifier could be used in the current approach which makes more use of linguistic information. For example,

by using dependency-based kernels (Bunescu and Mooney, 2005a; Kate, 2008) or syntactic kernels (Qian et al., 2008; Moschitti, 2009) or by including the word categories and their POS tags in the subsequences. Also, it will be interesting to see if a kernel that computes the similarity between subcard-pyramids could be developed and used for relation classification.

7 Conclusions

We introduced a card-pyramid graph structure and presented a new method for jointly extracting entities and their relations from a sentence using it. A card-pyramid compactly encodes the entities and relations in a sentence thus reducing the joint extraction task to jointly labeling its nodes. We presented an efficient parsing algorithm for jointly labeling a card-pyramid using dynamic programming and beam search. The experiments demonstrated the benefit of our joint extraction method over a pipelined approach.

Acknowledgments

This research was funded by Air Force Contract FA8750-09-C-0172 under the DARPA Machine Reading Program.

References

Razvan C. Bunescu and Raymond J. Mooney. 2005a. A shortest path dependency kernel for relation extraction. In *Proc. of the Human Language Technology Conf. and Conf. on Empirical Methods in Natural Language Processing (HLT/EMNLP-05)*, pages 724–731, Vancouver, BC, October.

Razvan C. Bunescu and Raymond J. Mooney. 2005b. Subsequence kernels for relation extraction. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, Vancouver, BC.

Razvan Bunescu, Ruifang Ge, Rohit J. Kate, Edward M. Marcotte, Raymond J. Mooney, Arun Kumar Ramani, and Yuk Wah Wong. 2005. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine (special issue on Summarization and Information Extraction from Medical Documents)*, 33(2):139–155.

Andrew Carlson, Justin Betteridge, Estevam R. Hruschka, and Tom M. Mitchell. 2009. Coupling semi-supervised learning of categories and relations. In *SemiSupLearn '09: Proceedings of the NAACL HLT 2009 Workshop on Semi-Supervised Learning for Natural Language Processing*, pages 1–9, Boulder, Colorado.

Michael Collins. 2002. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, pages 489–496, Philadelphia, PA.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. 2007. Relation extraction and the influence of automatic named-entity recognition. *ACM Trans. Speech Lang. Process.*, 5(1):1–26.

D. Jurafsky and J. H. Martin. 2008. *Speech and Language Processing: An Introduction to Natural Lan guage Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Upper Saddle River, NJ.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proc. of the 21st Intl. Conf. on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pages 913–920, Sydney, Australia, July.

Rohit J. Kate. 2008. A dependency-based word subsequence kernel. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, pages 400–409, Waikiki,Honolulu,Hawaii, October.

Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge, MA.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Scott Miller, Heidi Fox, Lance A. Ramshaw, and Ralph M. Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *Proc. of the Meeting of the N. American Association for Computational Linguistics*, pages 226–233, Seattle, Washington.

Alessandro Moschitti. 2009. Syntactic and semantic kernels for short text pair categorization. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 576–584, Athens,Greece, March.

John C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Alexander J. Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 185–208. MIT Press.

Vasin Punyakanok and Dan Roth. 2001. The use of classifiers in sequential inference. In *Advances in Neural Information Processing Systems 13*.

Longhua Qian, Guodong Zhou, Fang Kong, Qiaoming Zhu, and Peide Qian. 2008. Exploiting constituent dependencies for tree kernel-based semantic relation extraction. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 697–704, Manchester, UK, August.

Sebastian Riedel, Hong-Woo Chun, Toshihisa Takagi, and Jun'ichi Tsujii. 2009. A Markov logic approach to bio-molecular event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 41–49, Boulder, Colorado, June. Association for Computational Linguistics.

D. Roth and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proc. of 8th Conf. on Computational Natural Language Learning (CoNLL-2004)*, pages 1–8, Boston, MA.

D. Roth and W. Yih. 2007. Global inference for entity and relation identification via a linear programming formulation. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*, pages 553–580. The MIT Press, Cambridge, MA.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of 7th Conf. on Computational Natural Language Learning (CoNLL-2003)*, Edmonton, Canada.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

Min Zhang, Jie Zhang, Jian Su, and Guodong Zhou. 2006. A composite kernel to extract relations between entities with both flat and structured features. In *Proc. of the 21st Intl. Conf. on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, Sydney, Australia, July.