Classifier Based Machine Comprehension

Qiaojing Yan
SU ID / 06000158
qiaojing@stanford.edu

Yixin Wang
SU ID / 06047182
wyixin@stanford.edu

Abstract

In this report we implement a machine comprehension system, and train and test it on the MCTest dataset (Richardson et al., 2013). We treat it as a classification problem. We use baseline features and syntactic features to compute the score for each candidate answers. We also used a set of NLP techniques, including word embedding, coreference resolution and lemmatization to improve the performance of our features. We train our system using a max-margin loss function with a latent variable. Our result shows significant improvement over the original baseline.

1 Introduction

To understand text as well as human is one major goal of natural language processing. Traditionally, Turing test had been used to measure the intelligence of an AI system. However, it over emphasize the system's behaviorial similarity compared to human. Other tasks had been proposed by the academic community to measure the progress of machine comprehension, such as information extraction, relation extraction, semantic parsing and textual entailment. However, these techniques are evaluated individually and does not show clearly how much we advance towards real text understanding. To provide a better measurement of NLP intelligent system performance, Richardson et al. (2013) proposed a MCTest, which tests question answering based on short passages. The passages are fictional stories and their complexities are limited to those that a young child could understand. A related test, bAbI, had been proposed by Weston et al. (2015). They proposed a set of prerequisite toy tasks which should be solved before any NLP system can truly understand text. In this paper, we focus on the MCTest, and try to build a classifier machine comprehension system to do it.

2 Related Work

In the paper introducing the MCTest, Richardson et al. (2013) showed two baseline algorithms. The first baseline used a sliding window and they matched the words in the window with the bag of words constructed from the question and a candidate answer. To take into account long range dependencies, they added a word-distance based algorithm. Their simple baseline performed surprisingly well on the MCTest.

Smith et al. (2015) improved upon the baseline by building a multi-pass system which increase its window size on each pass, from 2 tokens to a length of 30 tokens. By carefully tuning the details of the system(For example, they do coreference resolution selectively.), they achieved an accuracy 5% higher than the baseline.

Sachan et al. (2015) assumed that there is a hidden structure between the question, correct answer, and text. They used a latent structural SVM (LSSVM) to learn the latent answer-entailing structures that helps answer questions about a text.

Wang et al. (2015) used a max-margin classifier to predict the correct answer. Their classifier incooperates syntactic features. They transform each question answer pair into a statement. They compared the dependency tree of the statement to each sentence in the passage. They also used frame semantics and word embeddings as features. They stated in their paper that their score is the highest among published results.

In this paper, we follow the work of Wang et al. (2015) and build a classifier based machine comprehension system. Our system focus on the questions that require only one sentence in the passage to answer. We incorporate both lexical features and syntactic features.

3 Task Description

MCTest is a multiple-choice question answering task on fictional stories written by human. The dataset is open-domain. However, the complexity of the story is restricted to concepts and words that a 7 year old could understand. One example is shown in figure 1. Each question has four candidate answers and there is only one correct answer. Before the question, "one" means the answer is dependent only on one sentence in the passage, while "multiple" means it needs to be deduced from multiple sentences. The dataset consists of two subsets, MC160 which has 160 stories, and MC500 with 500 stories. Each story has four questions. Each subset is divided into training set, development set and test set.

Billy went to the farm to buy some beef for his brother's birthday. When he arrived there he saw that all six of the cows were sad and had brown spots. The cows were all eating their breakfast in a big grassy meadow.

(i) One: What did Billy buy at the farm?
A) Beef
B) Chicken
C) Cows
D) Fence
(ii) Multiple:What color were the spots on the cows?
A) Blue
B) Brown
C) White
D) Black

Figure 1: One example from the MC Test

4 Classifier Model

For the MCTest, We denote each story text as a passage P, and W is the set of sentences in the passage. $w \in W$ denotes one sentence in this passage. The related questions q's each has four options denoted as $a_m (m = 1, 2, 3, 4)$. Following the work by Wang et el. (2015), we predict the answer $\hat{a}$ using:

$$\hat{a} = \arg \max_{a_m \in A} \max_{w \in W} \theta^T f(P, w, q, a_m) \quad (1)$$

In the equation, f is the feature extractor, and θ is the parameter that is to be learned during training. To learn the θ, we minimize a max-margin loss function that is l_2-regularized:

$$\min_{\theta} \lambda \|\theta\|^2 + \sum_{i=1}^{n} \left(- \max_{w \in W} \theta^T f(P^i, w, q^i, a^i) + \max_{a \in A} \left(\max_{w' \in W} \theta^T f(P^i, w', q^i, a) + \Delta(a, a^i) \right) \right) \quad (2)$$

In the equation, a^i is the correct answer. $\Delta(a, a^i) = 1$ if $a = a^i$, and equals 0 otherwise. w is a latent variable indicating which sentence in the passage the feature corresponds to. This optimization problem takes the form of structural SVM with latent variables. Because of this, gradient descent does not converge. During implementation we originally adopted the gradient descent approach and encountered some convergence problems, thus we revised the approach and took the Concave-Convex Procedure (CCCP) as described in (Yu and Joachims, 2009). In this case, it can be proved that CCCP is equivalent to the following procedure:

Algorithm 1 CCCP algorithm

procedure CCCP
  Initialize θ to random value;
  for iteration $t = 1, 2, ..., T$ **do**
    Step1 Holding θ as fixed,
    optimize the concave term:
    $\min_\theta \sum_{i=1}^{n} \left(- \max_{w \in W} \theta^T f(P^i, w, q^i, a^i) \right)$,
    finding optimum w;
    Step2 Holding w as fixed, optimize
    the convex term (which is the rest of
    this equation) using gradient descent,
    finding optimum θ;
  end for
end procedure

It can also be proved that in each update of w and θ, the objective function is decreasing, and thus it will converge eventually. We fine tune the hyper-parameter λ for regularization term on the development set. The parameter giving the best performance varies among feature sets, and will be discussed later.

5 Features

As stated in section 4, our feature extractor takes in (P^i, w, q^i, a) and produces a set of features. In our system, some features depend on w, which means they depend on a particular sentence in the passage, and are functions of (P^i, w, q^i, a). Other features do not depend on w, and are functions of (P^i, q^i, a^i).

5.1 Baseline Features

We use the baseline feature reported by Richardson et al. (2013). The algorithm has been explained in section 2. Their original feature was only a function of (P^i, q^i, a^i). In our work, we also use corresponding sliding window and distance features which set the window to a particular sentence in combination with their original feature. Our new feature are functions of (P^i, w, q^i, a).

5.1.1 Sliding window bag-of-words feature

This feature matches the bag-of-words feature of the question and proposed answer pair with the passage (2013), and the intuition is that the more match the better. Besides this, we have also implemented a slightly modified sliding-window bag-of-words feature, which is calculated in the same way as this feature, except that it is calculated sentence-by-sentence.

5.1.2 Distance punishment

This feature measures the minimum number of words between any words in question stem and any words in proposed answer, with stopwords removed. We used the stopword list provided by the MCTest database which contains 582 stopwords.
Similar to what we did with the sliding-window bag-of-words feature, we also implemented a sentence-by-sentence variation of distance punishment feature.

5.2 Syntactic Features

For syntactic features, we first convert each question answer pair into a statement. In figure 2 are some real examples. Then we compare the dependency tree of the statement to that of a sentence in the passage.

5.2.1 Statement

We use a rule based system to transform each question answer pair into a statement. The system work as follows:

Question:	Who is the character that is fat and is being bullied?
Choice:	Alex
Statement:	Alex is the character that is fat and is being bullied.
Question:	Why did James stand up to the bullies?
Choice:	He wanted to protect himself.
Statement:	James stand up to the bullies because He(he) wanted to protect himself.
Question:	What does the main character say her dad looks like on the ground?
Choice:	A bear
Statement:	the main character say A bear her dad looks like on the ground.

Figure 2: Example of converting question answer pairs into statements

(i) Labeling and parse the question sentence, find the wh-word c and the root word r in the collapsed dependency tree. The wh-word that our system covers are {what, who, why, when, how, where, which}.
(ii) Use a set of rules to transform question answer pair into statement. The original rules reported in (Wang et al., 2015) sometimes does not generate the correct statement. We improved upon their system and developed our rules:

- c = what, POS(r) $\in$ {VB, VBD, VBP}
  If dobj(r, c): Find word w that satisfies nsubj(r, w). Delete c and insert answer a after r.
  If nsubj(r, c): Replace c with a;
- c = what, POS(r) = WP
  Replace c with a.
- c = what, POS(r) = NN
  If nsubj(r, c): Replace c with a.
- c = which
  Delete the word after c. Replace c with a.
- c = where, POS(r) $\in$ {VB, VBP}
  If advmod(r, c): Find word w that satisfies dobj(r, w). Delete c and the word after it.

Insert a after w, or insert a after r if w is not found.

- c = where, POS(r) = NNP
  If advmod(r, c), delete c and the word after it. Insert a after r, and insert the word after c between r and a.

- c = who, POS(r) $\in$ {VB, VBD, VBG} If nsubj(r, c), replace c with a.

- c = who, POS(r) = WP
  Replace c with a.

- c = how many
  Replace c with a.

- c = how, POS(r) = VB
  Find word w that satisfies nsubj(r, w). Delete words before w. Insert answer a after w. Insert "to" between w and a.

- c = why
  Delete c and the word after it. Append answer a after the question sentence. Insert "because" between question sentence and a.

The above set of rules can transform a wide range of questions into statements. However, due to the variaties of the form of questions and answers, sometimes our StatementFactory produce gibberish, such as the last example in figure 2.

5.2.2 Feature

For syntactic feature, we use dependency parsing on both the statement and each statement in the passage. We count the common dependency arc that are used both in the statement and passage sentence as our feature. We denote each edge in the dependency tree as $r(u, v)$. Here u is the root word, v is a child word, and r is their dependency relation. Suppose the dependency tree of the statement has an edge $r(u, v)$, and the dependency tree of a sentence in the passage has $r'(u', v')$. These two edges are equal if $u = u'$, $v' = v$ and $r' = r$. For the syntactic feature, we count the number of edges that the statement and sentence both have.

5.3 Enhancing Features

To make our features better, we use a set of techniques to improve their performance.

5.3.1 Coreference Resolution

Because of the inherent nature of our classifier that it assumes the answer of a question is based on a single sentence in the passage, it is hard for our system to answer questions depending on multiple sentences. The intuition is that coreference resolution could bring in some information from other sentences to improve performance. We use the rule-based multi-pass coreference resolution system implemented in (Lee et al., 2011), and substitute words in the passage with the cluster center of a coreference chain. For example, the sentence "But before he did anything he had to fix the toilet." becomes "But before Tom did anything Tom had to fix the toilet." after substitution.

5.3.2 Lemmatization

Word in passage and word in question may have different forms. For example, "came" in a passage sentence "Soon her friend Lion came over", and "come" in the question "Who didn't come to the party?". So we use lemmatization to capture words that have the same base form.

5.3.3 Word Embeddings

To capture the meaning of the word and to recognize words that are close in meaning, we also use word vector to represent word. The word vector has dimensionality of 300. We calculate the inner product cos(word_1, word_2) between word vectors. If two words are same, their inner product is 1. Otherwise, it is less that 1. The inner product reflects how close two words are. For the sliding window feature, the sum of all word vectors in the window is v_s, and the sum of word vectors in the question and candidate answer is v_q. We use cos(v_s, v_q) as our feature.

5.3.4 Dealing with Negations

Sometimes the question has the form of negation. For example, the first question in figure 4. In this case, we need to negate the score of each option. Following the method of Wang et al. (2015), if a question contains "not" or "n't" and does not begin with "how" or why", we identify it as negation.

6 Experiments

We use the Stanford CoreNLP API (Manning et al., 2014) in our program to do
(1) tokenization and lemmatization,
(2) sentence splitting,
(3) part of speech tagging,

(4) constituent parsing,
(5) dependency parsing,
(6) coreference resolution.

To obtain the word vector, we use the pre-trained word embedding data from the word2vec site[1].This set of pre-trained word vectors has a dimensionality of 300.

Our source code can be found on github[2]. The `TaskReader` class object reads both the training and development dataset. Then the training set and correct answers are fed into a `ClassiferBased` MC system to train the parameters. After that, development set are fed to the trained classifier and predicted answers are produced. The classifier has a list of `Featurizer` objects, which is used to derive features. The `StatementFactory` class is used to convert question answer pair to statement, which is then used in `SyntacticFeaturizer`. The `WordEmbeddingDict` is used to find the word vectors.

6.1 Training

For training, we combined the data in MC160 training set and MC500 training set. We then tune our parameters using the MC160 and MC500 development set. The final test is performed on the MC160 test set and MC500 test set. We fine tuned parameters on the dev set. The learning rate $\alpha = 0.0001$.

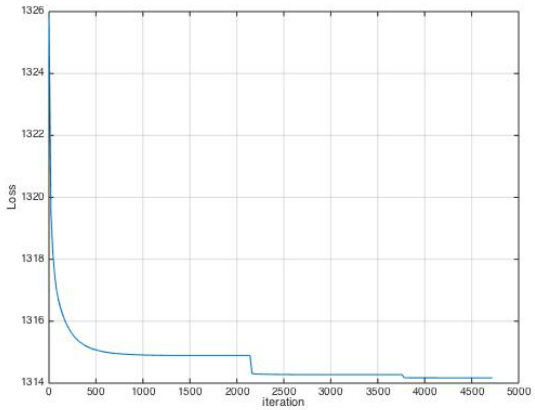

Figure 3: Minimization of Loss function as the number of iteration times grows during training, until convergence

Figure 3 illustrates the decrease of loss function during training. There are some discontinuities as the loss reduces. These discontinuities are yielded by *step 1* of Algorithm 1, i.e. optimizing the concave term. The CCCP algorithm guarantees that loss function is decreasing in every step until convergence.

7 Analysis

In this section, we describe the accuracy of different feature combinations on development and test set, then we do error analysis to see how each feature works.
We denote sliding window bag-of-words feature as B, distance punishment as D, syntactic features as S, bag-of-words similarity measured by word vector cosine similarity as B^w, sentence-by-sentence bag-of-words feature as B^s, sentence-by-sentence distance feature as D^s, bag-of-words feature with lemmatization as B^l, and distance feature with lemmatization as D^l.

7.1 Performance result summary

The performance of different feature sets on development and test set is listed below. "One" refers to the performance on questions that depend on one sentence, "Multiple" refers to the performance on questions that depend on multiple sentences, and "All" refers to the performance on the entire set.

7.2 Successes

7.2.1 Feature Set 1

Feature set: $\{B,D\}$
With bag-of-words and distance punishment features, the accuracy is 0.5781 on dev set, and 0.5333 on test set($\lambda = 0.1$). The feature weights are $[0.7431, -0.3298]$ for B and D features, respectively, which makes perfect sense because for B feature, the larger the matching set, the more likely it is to be the desired answer, and thus it bears a positive weight. On the other hand, for D feature, the fact that a question word and an answer word have a large distance in the passage is not a good sign because it usually indicates that they are less related, and thus it bears a negative weight.
Feature set 1 can answer many easy questions right.
e.g.
what was Sammy barking at?
A.a dog; B.a cat; C.a bug; D.a toad"

[1]https://code.google.com/p/word2vec/
[2]https://github.com/qiaojingy/MC

Feature Set	One		Multiple		All	
	dev	**test**	**dev**	**test**	**dev**	**test**
$B + D$	0.6259	0.5390	0.5414	0.5328	0.5781	0.5333
$B^l + D^l$	0.5899	0.5964	0.5690	0.5373	0.5781	0.5643
$B^l + D^l + S$	0.6115	0.6250	0.5912	0.5548	0.6000	0.5857
$B^l + D^l + S + B^w$	0.6187	0.6224	**0.6022**	0.5417	0.6093	0.5786
$B^l + D^l + S + B^s + D^s$	0.6403	**0.6588**	0.5525	**0.5636**	0.5906	**0.6071**
$B + D + S + B^l + D^l + B^w$	0.6331	0.6250	0.6022	0.5504	**0.6156**	0.5845
$B^l + D^l + S + B^s + D^s + B^w$	**0.6536**	0.6380	0.5570	0.5329	0.6012	0.5810

Table 1: ClassifierBased perf. on dev and test set

(gold answer: D)
This question corresponds to the sentences in the passage:
"When they got to the fishing hole Sammy ran over to a rock and started barking with his hair standing up. Tommy ran to see what Sammy had found. Under the edge of the rock was a huge green toad".
The matching of "barking" and "toad" makes answer D more likely, however, the word "dog" also appeared in the passage, making answer A more likely. But because the word distance from "dog" to "bark" is farther than that of "bark" and "toad", the distance punishment pulls the score of answer A down, reavealing the correct answer. However, this baseline feature set has lots of problems, which will be discussed later as we add more features.

7.2.2 Feature Set 2

Feature set: $\{B^l, D^l\}$
Feature set 2 uses the same set of features as feature set 1, but uses lemmatization to cope with the issue of mismatch of two words because of word forms when they are indeed the same word if converted to base form. The accuracy is 0.5781 on the dev set and 0.5643 on the test set ($\lambda = 0.1$). The weights of B^l and D^l features are $[0.8777, -0.1798]$ respectively, which makes sense based on the same reasons as described in section 7.1.
Feature set 2 works better than set 1 when there is a change in word form.
e.g. What color are our plates?
A.red; B.white; C.blue; D.green"
(gold answer: C)
This question corresponds to the sentence in the passage "Each plate is blue". Feature set 1 gives the incorrect answer B. It regards "plate" and "plates" as totally different words, and because both "white" and "blue" appeared in the passage, it picks "white" which appears earlier. However, Feature set 2 is cleverer on this, for it knows "plate" and "plates" are the same by comparing them in base form, thus reaching the correct answer.

7.2.3 Feature Set 3

Feature set: $\{B^l, D^l, S\}$
Feature set 3 adds syntactic feature which captures the structure of sentences. the accuracy is 0.6000 on the dev set and 0.5857 on the test set ($\lambda = 1$). The weights of B^l, D^l and S features are $[0.7743, -0.1928, 0.1445]$ respectively, which makes sense because the more similar the stem-option statement is in dependency structure to an original sentence in passage, the more likely it is the correct option.
Feature set 3 works better than set 2 especially when there are "wh-" questions of which the response is a sentence.
e.g. Why did the boy pick an instrument?
A.His brother played an instrument;
B.He thought guitars were cool;
C.His parents thought it was important;
D.He wanted to join the school band.
(gold answer: C)
This question corresponds to the sentence in the passage "His parents wanted him to pick a good one because playing an instrument was very important to them", the `StatementFactory` class returns a new statement which gets rid of word "why" and glues stem with options adding the word "because". The new statement is similar to the corresponding sentence by both having prepositional "because" arc, auxpass arc with "is", etc. Thus system with Feature set 3 was able to return the correct answer C. On the other hand, fea-

ture set 2 incorrectly returned D because the word "band" frequently appeared in the passage. In this type of cases as described above, syntactic features can improve accuracy.

7.2.4 Feature Set 4

Feature set: $\{B^l, D^l, S, B^w\}$
Feature set 4 uses word embedding, and adds measurements of word distance using the cosine similarity of two vectors so as to better compare between words. The accuracy is 0.6156 on the dev set and 0.5786 on the test set ($\lambda = 10$). The weight of B^l,D^l,S,B^w features are $[0.6996, -0.1434, 0.1454, 0.46426, 0.0344]$ respectively (where B^w feature has 2 dimensions), which makes perfect sense because for B^w feature, the closer the stem-option summation vector is to the target sentence summation vector, the better, and for other features, the reasons has been discussed above.
Word vector embedding enables us to measure the distance in meaning between two words. It models *meaning* better than any other features here, and thus it can capture some surprisingly subtle implications in comprehension, as is shown in the example below.
*e.g.*Did Ashley and Ethan catch any fish?If so how many?
A.Two; B.One; C.Three; D.No
(gold answer: D)
Surprisingly, Feature Set 4 is able to find the exact corresponding sentence in passage "$w = 19$: Even if they didn't catch anything", and return the correct answer! This might be because "anything" and "no" are close to each other in terms of *word vector similarity*, but are completely different words for other features. In this example, Feature set 3, which does not have word embedding, returns some random sentences "$w = 0$: The day was sunny and warm". However, in general word embedding feature has caused slight decrease in accuracy, which might be caused by some confusion caused by word vector.

7.2.5 Latent variable w

The latent variable w in our classifier model denotes which sentence in the passage can tell us the answer to a question, i.e., the sentence in the passage that a stem-option pair correspond to, and during testing, we found that our system is able to find the right sentence in many cases. Because only Syntactic feature S is relevant to finding optimum w.
e.g. The girl and her dog went walking into what?
A.A field; B.The woods; C.A park; D.A garden
(gold answer: D)
Sentence found: $w = 1$: The young girl and her dog set out a trip into the woods one day.
This indicates that syntactic feature, as well as B^s and D_w features have been successful in capturing the similarity between passage sentence and question answer pair.

7.3 Failures

When dealing with dev and test set of MCTest dataset, we found that the test set is generally more difficult than the dev set, and that some features that has brought considerable improvement of performance on the dev set could even lead to depreciation of performance on the test set.

7.3.1 Coreference Resolution failure analysis

We found out that adding coreference resolution decreases accuracy by around 5% for almost every set of features. This might be because coreference resolution is not always correct, and that its mistakes might have cascaded to mistakes of dependency parsing used by syntactic feature S and word distance feature D.

7.3.2 Systematic Errors

Our design does not have sepcial technics to deal with multisentence reasoning. To answer questions that requires multiple sentences in the passage, the system needs to have a memory, and a working model of the world. For example, consider the passage and question in figure 4. To answer the second question in figure 4, the system needs to have a "sense" of time sequence and causality. Also, it also needs to remember the sequence of events that has happened. It even needs to know that we eat cake at birthday.

It was Jessie Bear's birthday. She was having a party. She asked her two best friends to come to the party. She made a big cake, and hung up some balloons.
Soon her friend Lion came over. Then her friend Tiger came over. Lion and Tiger brought presents with them.

(i) Who didn't come to the party?
A) Lion
B) Tiger
C) Snake
D) Jessie Bear
(ii) How did Jessie get ready for the party?
A) made cake and juice.
B) made cake and hung balloons.
C) made juice and cookies.
D) made juice and hung balloons.

Figure 4: Another example in the MC Test

8 Discussion and Future Work

As discussed in section 7.3.2, to truly understand the text, a machine comprehension system needs to have a memory component and a basic model of the world.

A number of researchers tried in this direction. Narasimhan and Barzilay (2015) capture discourse information by modeling relation between sentence. However, their system does not keep memory of want happened and it can only keep track of at most two sentences at the same time. Weston et al. (2015) used memory network to keep a memory of the world and capture discourse relation. Though their model showed success showed success on the bAbI tasks, these tasks are artificially sythesised and each tasks are usually less than five sentences. Memory model currently does not perform well on human written texts such as in the MCTest. For the word model, some effort is done by Wang et al. (Wang et al., 2015) to incorporate frame semantics. However, this is far from a good world model. So building a true machine comprehension system is currently still a great challenge. The author thinks building a more robust memory network on bAbI test may be a good start point.

References

Matthew Richardson, Christopher JC Burges, Erin Renshaw 2013. *MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text.* EMNLP. Vol. 1. 2013.

Jason Weston, Antoine Bordes, Sumit Chopra, Tomas Mikolov 2015. *Towards AI-complete question answering: a set of prerequisite toy tasks.* arXiv preprint arXiv:1502.05698 (2015).

Karthik Narasimhan, Regina Barzilay 2015. *Machine comprehension with discourse relations.* 53rd Annual Meeting of the Association for Computational Linguistics. 2015.

Hai Wang, Mohit Bansal, Kevin Gimpel, David McAllester 2015. *Machine comprehension with Syntax, Frames, and Semantics.* In Proceedings of ACL: Short papers (2015): 700.

Mrinmaya Sachan, Avinava Dubey, Eric P.Xing, Matthew Richardson 2015. *Learning answer-entailing structures for machine comprehension.* In Proceedings of ACL (2015).

Smith Ellery, Nicola Greco, Matko Bosnak, Andreas Vlachos 2015. *A Strong Lexical Matching Method for the Machine Comprehension Test.* In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 16931698.

Marie-Catherine De Marneffe, Christopher D. Manning 2008. *Stanford typed dependencies manual.* In Technical report, Stanford University, 2008.

Christopher Manning, Mihai Surdeanu, Johb Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. *The Stanford CoreNLP natural language processing toolkit.* In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60. 2014.

Chun-Nam John Yu and Thorsten oachims. 2009. *Learning structural SVMs with latent variables.* In Proceedings of the International Conference on Machine Learning, pp. 18, 2009.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathaneal Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. *Stanfords Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task.* In Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task. Association for Computational Linguistics, pp. 2834, 2011.