Learning How to Cluster
With Application to Coreference Resolution

Jason D. M. Rennie
jrennie@csail.mit.edu

August 13, 2004

1 The Problem

We consider the problem of co-reference resolution. We assume that noun phrases have already been extracted; let's call each NP a "mention." What is left is the task of determining which mentions refer to the same thing. We assume that we have a training corpus where this has already been done. Let $\{(x_1, y_1), \ldots, (x_l, y_l)\}$ be a set of l mentions and their "labels;" the label of an example is nothing more than an identifier—examples x_i and x_j refer to the same entity iff $y_i = y_j$. Given a new text and extracted mentions, $\{x_{l+1}, \ldots, x_{l+u}\}$, our goal is to provide label identifiers so that $y_i = y_j$ iff x_i and x_j refer to the same entity.

2 The Idea

Co-reference resolution can be thought of as a clustering problem: the task is to group together mentions that refer to the same thing. But unlike more traditional clustering problems like document clustering, similarity is asymmetrical. Consider this example:

> President Bush addressed the crowd. "We must fortify our country against terrorism," he said.

"He" refers to "President Bush," not the other way around. Reference is usually uni-directional. One reference refers to another; a chain of references ultimately refer to a root reference that provides the full name of a named entity. We use this as the basis for our model: each reference chain is identified by a single, "root" mention.

3 The Game

We formulate the co-reference resolution problem as a game. As in many reinforcment learning problems, the goal is to maximize expected reward. But,

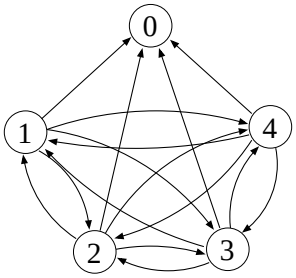

Figure 1: An example graph.

unlike most reinforcement learning problems, we will not be selecting actions; rather, we will be designing a "world" so that some set of actions (out of our control) maximize reward. Solving the game corresponds to training a model for co-reference resolution.

Our "world" is a collection of graphs. There is one graph for each cluster in our training data, $\{1, \ldots, n\}$. Each graph has one vertex per example, plus one special "zero" vertex. There is an edge from each non-zero vertex to every other vertex. See Figure 1 for a graphical depiction. Each vertex represents a state and each edge represents a transition. State zero is an end/termination state. Associated with each edge is a transition probability. Transition probabilities are the same for all n graphs. In each graph, there is a special non-zero vertex, the "root," such that if a transition is made from the root to the zero vertex, we gain a reward of 1. Other transitions yield no reward.

The game is as follows. Set transition probabilities and roots so as to maximize total expected reward. Each example, x_i, starts at node i in graph y_i and follows a random walk according to the transition probabilities. Our total expected reward is the sum of expected rewards for each example.

4 A Comment on Mechanism Design

This game can be related to the Economics problem of mechanism design. The mechanism design problem is as follows. Two (or more) parties would like to create an agreement to benefit the involved parties. The goal is to design an agreement that will be mutually beneficial and not encourage the parties to lie or take advantage of each other. In some ways, our game is similar. Designing a contract can be compared to determining the transition probabilities and roots in our game. For each configuration, we can exactly determine the total expected reward; likewise for the mechanism design problem, exact payoff can be calculated for a given contract. However, mechanism design involves a trade-off between multiple parties; our game is not between multiple parties, but similar to mechanism design, there is no single "correct" objective.

5 Roots

Define the probability of a random walk staring at i making a $j \to 0$ transition to be

$$p_i(j) = q_{i0}\delta_i(j) + \sum_k q_{ik} p_k(j), \tag{1}$$

where $\delta_i(j)$ equals one if i equals j, zero otherwise, and q_{ab} is the probability of transitioning from state a to state b. Note that if the $\{q_{ab}\}$ are specified, we can directly retrieve the probabilities via matrix inversion. The vectorized recursive equation is:

$$\vec{p}(j) = \vec{q}_0\vec{\delta}(j) + Q\vec{p}(j). \tag{2}$$

The solution is

$$\vec{p}(j) = \vec{q}_0\vec{\delta}(j)(I - Q)^{-1}. \tag{3}$$

We say that the most likely zero-transition for a state is its "root." The root for state i is $r^*(i) = \arg\max_j p_i(j)$.

6 Generalization

So far, we have assumed that roots and transition probabilities are given. We have also focused on the training data. For this framework to be useful for supervised learning, we must be able to learn roots and transition probabilities and generalize to unseen data.

Let $r(y)$ be the root for graph (label) y. Total expected reward (TER) can be written as

$$\sum_{i=1}^{l} p_i(r(y_i)). \tag{4}$$

TER is an objective that can be used to determine transition probabilities and roots. But, the problem is underconstrained; without constraints on the transition probabilities, TER can be trivially maximized.

We now constrain the transition probabilities. Define

$$q_{i0} = \frac{1}{Z_i}\exp(w_0), \quad \text{and,} \tag{5}$$

$$q_{ij} = \frac{1}{Z_i}\exp(s(i,j)), \tag{6}$$

Where $Z_i = \sum_{j=0}^{n} q_{ij}$ is the normalization constant and $s(i,j)$ is the similarity between examples i and j. Similarity is defined as:

$$s(i,j) = \sum_k w_k f_k(i,j). \tag{7}$$

where the $\{f_k\}$ are features on pairs of examples and the $\{w_k\}$ are weights that provide a metric on the feature space. For the problem of co-reference resolution, the $\{f_k\}$ should be defined so that, given appropriate weights, $s(i,j)$ is large if i refers to j and $s(i,j)$ is small if i does not refer to j.

Given a set of training data, $\{(x_1,y_1),\ldots,(x_l,y_l)\}$, we learn weights to maximize TER. But, since the training data does not specify roots, only which examples are in the same cluster, we must use a relaxed form of TER that learns the root for each cluster. Let r_{yi} represent the weight of example i as the root of cluster y. We define the relaxed TER objective as:

$$\sum_{i=1}^{l} \sum_{j:y_j=y_i} \frac{\exp(r_{yj})}{Z_y} p_i(j), \tag{8}$$

where $Z_y = \sum_j \exp(r_{yj})$ is the normalization constant. $\frac{\exp(r_{yj})}{Z_y}$ can be thought of the probability that example j is the root for graph y. By maximizing the relaxed TER objective, we learn roots for the training data; we also learn the weights, $\{w_k\}$, that allow us to generalize similarity to new data. Given a set of trained weights, $\{w_k\}$, we use Equation 3 to solve for root probabilities, then we assign two examples, x_i and x_j, to the same cluster if they have the same roots ($r^*(i) = r^*(j)$).

7 Co-Reference Resolution and Consistency

The root probability definition as we have defined it is recursive. This is inappropriate for the problem of co-reference resolution—nearly all refernces are back-references. Identifying the occasional forward-reference may best be treated as a separate task. To restrict our model to back-references, we alter $p_i(j)$ so that the sum is only over $k < i$:

$$p_i(j) = q_{i0}\delta_i(j) + \sum_{k<i} q_{ik} p_k(j). \tag{9}$$

The normalization terms for the q's are redefined appropriately: $Z_i = \sum_{j=0}^{i-1} q_{ij}$. There are additional advantages to this model: (1) the $\{p_i(j)\}$ can be solved via dynamic programming (matrix inversion is unnecessary), and (2) we can show that determining roots $\{r^*(i)\}$ for a set of test data is "consistent." By consistent, we mean that $r^*(r^*(i)) = r^*(i)\ \forall i$; that is, the root of any example is a self-root. Thus, in labeling testing data according to roots, a root is always part of the same cluster as examples that point to it.

7.1 Consistency Proof

We show something slightly stronger than discussed. Note that $p_i(r^*(i)) \geq p_i(j)\ \forall j$.

Theorem 1. *Let $i < j < k$. $p_j(i) > p_j(j) \Rightarrow p_k(i) > p_k(j)$.*

Proof.

$$p_k(i) \geq p_j(i) \sum_{p:\text{path } k \to j} \prod_{(a,b) \in p} q_{ab} > p_j(j) \sum_{p:\text{path } k \to j} \prod_{(a,b) \in p} q_{ab} = p_k(j). \quad (10)$$

□

Thus, each reference chain has a unique "root."

8 Convexity

Note that the (relaxed) TER objective is not convex. We can write $p_i(j)$ as a weighted sum of delta functions; if the coefficients of all the delta functions were log-linear, a convex objective would be possible. However, the coefficients are sums of log-linear terms. The normalization constant is a sum of log-linear terms. Generally speaking, ratios of convex functions are not convex.

9 Alternate Objective Function

Our TER objective function can be compared to a minimization of classification errors for a classificaiton problem. However, this objective is, in principle, not convex. Most classification algorithms use a bound on the classificaiton error, such as logistic loss or hinge loss. Reward in this problem is equivalent to classification accuracy. We achieve a bound on classification error by taking the log of the product of the rewards:

$$\log \prod_{i=1}^{l} p_i(r(y_i)) = \sum_{i=1}^{l} \log p_i(r(y_i)). \quad (11)$$