Cats have Tails

1 Search Problem

We run a search over mutations over specific words of the input candidate consequent. The search problem is given a consequent as input. This is composed of the following:

- A sentence, tokenized into coherent logical entities. For instance, *dog* and *cat* would be entities; but, so would *George W. Bush* – despite the latter covering multiple words.
- A monotonicity marking for each of these words. This is one of: upward monotone, downward monotone, or flat (no monotonicity information could be gathered).
- The relevant sense of the word. This is a WordNet synset when it's available; if not, this is set to a special sense denoting no synset. If a WordNet synset is available, it will never be set to the 'no synset' category.

The paths from the result of this searches are used as features in the learning problem (see Section 2).

We define the problem by specifying the state space, the valid transitions, along with the weights of the transitions. The algorithm used is a Uniform Cost Search, leading to the additional challenge of ensuring that the cost of each transition is very fast – that is, on the order of microseconds to keep pace with the memory accesses for the search.

1.1 State

A state in the graph is defined as a partial path from the consequent to a valid antecedent. That is, a state is primarily parameterized by its candidate fact which may be in the knowledge base. In addition, some auxilliary information is also tracked in a search state:

- The fact this fact mutated from. This is necessary to reconstruct the final path.

- The monotonicity marking of each word in the fact. This is necessary for the weights, to ensure that monotonicity is respected.
- The synset of each word in the fact. This is necessary to compute the valid mutations of that edge, according to the WordNet hierarchy.
- A bitmask denoting whether a word has been modified in the past. This is set when a word has been modified, and then another word is subsequently modified. This ensures that edits on a word are contiguous. We do not edit a particualr word, and then edit another word, and then perform more edits on the initial word.

1.2 Weights

TODO: This section is now wrong We can denote a state transition as (n_{-1}, n) – corresponding to a state n and the previous state $n-1$. The weight of a transition from (n_{-1}, n) to (n, n_{+1}) is given by appealing to the transitions between n_{-1} and n, and between n and n_{+1}.

We denote the type of transition (e.g., ins(w) or freebase(r)) between n_{-1} and n to be ϕ_{-1}, and the type of transition between n and n_{+1} to be ϕ_0. For both of these, we define weights $w_{\phi_{-1}}$ and w_{ϕ_0}, as well as a weight for the bigram of the two: w_{ϕ_{-1},ϕ_0}. Lastly, we incur a cost for exiting the domain of natural logic entailments: w_{nle}. Our cost is thus given by:

$$\text{cost} = -w_{\phi_0} - w_{\phi_{-1},\phi_0} - \mathbb{1}(\text{broke entailment}) \cdot w_{nle} \tag{1}$$

We constrain $w_\phi \leq 0$ to ensure that the cost is always positive.

Note, furthermore, that this is a natural decomposition of a log-linear model where ϕ denote features and w denote the weights of the features. The cost of a path will become:

$$\sum_i cost_i = -\sum_i \left[w_{\phi_i)} + w_{\phi_{i-1},\phi_i} \right] \tag{2}$$

If we exponentiate the negative of this, we arrive at:

$$\exp(\sum_i cost_i) = \exp\left(\sum_i \left[w_{\phi_i)} + w_{\phi_{i-1},\phi_i} \right] \right) \tag{3}$$

Over two classes: *true* and *false*; and, defining notation where ϕ denotes the vector of active path types taken and w denotes the global weight vector, we arrive at our log-linear model:

$$P(\textit{true}) \propto e^{w^T \phi} \tag{4}$$

This decomposition is important, as it allows our search to get smarter along with our learning algorithm, and allows us to find better support for facts as we learn what patterns entail "good" support.

1.3 Transitions

A valid transition is one between states (n_{-1}, n) and (n, n_{+1}), such that in relation to n, n_{+1} is one of:

- **Add/Remove a quantifier**. This is a class of transitions denoted by *ins(w)* or *del(w)* where w is the word being added or removed. For example, transitioning from *cat* to *a cat* or *every cat* or visa versa.
- **Add/Remove an adjective**. Similar to above, but with adjectives. These are distinguished from the above in that they have meaning for natural logic entailment. **TODO:** how do we make this efficient?
- **WordNet hypernymy**. This is a class of transitions denoted by *hyper* or *hypo*, For example, we could transition from *cat* to *feline* and eventually to *animal*
- **WordNet relations**. This is a class of transitions related to the other WordNet relations (e.g., antonymy), primarily to capture some of the inferences in natlog.
- **Freebase relations**. This is a class of transitions aimed primarily for proper nouns, traversing Freebase relations. These are denoted by *freebase(r)*, where r denotes the freebase relation we have traversed. For example, *Barack Obama* could transition to *USA* via the *employee_of* relation.
- **ReVerb relation**. As in the Freebase case, we can move along a ReVerb relation, denoted as *reverb(r)*.
- **Acronym and de-acronym**. Expanding or creating an acronym,denoted by *acronym* and *deacronym*.
- **Nearest neighbor similarity**. The fallback is to search for nearest neighbors in similarity space. This is, roughly, the equivalent of the CoNLL paper.
- **Drop sense**. Drop the sense of a word – this is necessary to begin operating in nearest neighbors space, or even with Freebase, etc. relations.
- **Infer sense**. Go from a sense-less word to one of its word senses. For example, chosing a particular sense for *cat* from the initially senseless definition.

2 Learning Problem

The prediction problem is a binary prediction task: is the given fact true or false, given a database of known facts. We divide this section into the model, the objective function, and the data (or lack thereof).

2.1 Model

Given a query fact (a_1, r, a_2) we define the **support** for that fact as the set of facts $\{(a'_1, r', a'_2)\}$ such that we have paths from $a_1 \rightarrow a'_1$, $r \rightarrow n'$, and $a_2 \rightarrow a'_2$. We are thus given a set of triples of paths $\{p_{a_1}, p_r, p_{a_2}\}$. When featurized,

2.2 Objective

3 Misc Snippets

3.1 Generalization of JC Similarity

Let us assume we have words w_1 and w_2, with a least common subsumer lcs. The JC distance $\text{dist}_{\text{jc}}(w_1, w_2)$ is:

$$\text{dist}_{\text{jc}}(w_1, w_2) = \log \frac{p(\text{lcs})^2}{p(w_1)p(w_2)} \tag{5}$$

We show that our search over the Wordnet hierarchy generalizes this similarity. In particular, let us define two features, $\phi_\uparrow$ and $\phi_\downarrow$, corresponding to going up and down the WordNet hierarchy, respectively. Traversing the Wordnet hierarchy from words $w \rightarrow w'$ thus fires the ϕ features with counts:

$$\phi_\uparrow(w \rightarrow w') = \log \frac{p(w')}{p(w)} = \log p(w') - \log p(w) \tag{6}$$

$$\phi_\downarrow(w \rightarrow w') = \log \frac{p(w)}{p(w')} = \log p(w) - \log p(w') \tag{7}$$

We now introduce weights associated with each of these two operations, denoted $\theta_\uparrow$ and $\theta_\downarrow$, for each pair of words participating in a WordNet edge. The score of a path is then defined as the dot product of the weights and features as described above: $\theta^{\text{T}}\phi$.

We can factorize this along the path $w_1, w_1^{(1)}, w_1^{(2)}, \ldots, \text{lcs}, \ldots, w_2^{(2)}, w_2^{(1)}, w_2$ as follows:

$$\begin{aligned}
\theta^{\text{T}}\phi = {} & \theta_\uparrow \left(\left[\log p(w_1^{(1)}) - \log p(w_1) \right] + \cdots + \left[\log p(w_1^{(n)}) - \log p(\text{lcs}) \right] \right) + \\
& \theta_\downarrow \left(\left[\log p(\text{lcs}) - \log p(w_1^{(n)}) \right] + \cdots + \left[\log p(w_1) - \log p(w_1^{(1)}) \right] \right) \\
& = \theta_\uparrow \left(\log \frac{p(\text{lcs}}{p(w_1)} \right) + \theta_\downarrow \left(\log \frac{p(\text{lcs}}{p(w_2)} \right) \\
& = \log \frac{p(\text{lcs})^{\theta_\uparrow + \theta_\downarrow}}{p(w_1)^{\theta_\uparrow} + p(w_2)^{\theta_\downarrow}}
\end{aligned}$$

Note that setting both $\theta_\uparrow$ and $\theta_\downarrow$ to 1 exactly yield JC similarity.