ASSIGNMENT 3: [COMP550]

JOSEPH D. VIVIANO

1. Lambda Calculus and Compositional Semantics

1.1. **a.**

$$\begin{aligned}(\lambda x.xx)(\lambda y.yx)z =&(\lambda y.yx)(\lambda y.yx)z\\ =&(\lambda y.yx)xz\\ =&xxz\end{aligned} \tag{1.1}$$

$$\begin{aligned}(\lambda uvw.wvu)aa(\lambda pq.q)z =&(\lambda pq.q)aa\\ =&a\end{aligned} \tag{1.2}$$

$$\begin{aligned}[(\lambda v.vv)(\lambda u.u)][(\lambda v.v)(\lambda v.w)] =&[(\lambda u.u)(\lambda u.u)](\lambda v.w)\\ =&(\lambda u.u)(\lambda v.w)\\ =&(\lambda v.w)\end{aligned} \tag{1.3}$$

1.2. **b.** : `DET -> no`, and `V -> hates`

No says that none of the objects with attribute P also have attribute Q, or "for all x, such that x is also P, x is not Q."

$$\lambda P.\lambda Q.\forall x.P(x) \rightarrow \neg Q(x) \tag{1.4}$$

Hates, a verb, denotes an event e:*something is hated.* This requires a *hater* z at said event, and a *hatee* x at said event:

$$\lambda w.\lambda z.w[\forall x.\exists e\ hates(e) \wedge hater(e,z) \wedge hatee(e,x)] \tag{1.5}$$

1.2.1. *A parsing tree.* Let `A` $\equiv \lambda z.\exists e.\ hates(e) \wedge hater(e,z) \wedge hatee(e, COMP550)$

The semantics of S are as follows:

Date: Sept 2018.

(1.6)
$\lambda Q.\forall x.student(x) \rightarrow \neg Q(x)](A)$
$=\lambda Q.\forall x.student(x) \rightarrow \neg A(x)$
$=\lambda Q.\forall x.student(x) \rightarrow \neg[\lambda z.\exists e.\ hates(e) \wedge hater(e,z) \wedge hatee(e, COMP550)](x)$
$=\lambda Q.\forall x.student(x) \rightarrow \neg[\exists e.\ hates(e) \wedge hater(e,X) \wedge hatee(e, COMP550)$

`S` $< \lambda Q.\forall x.student(x) \rightarrow \neg\exists e.\ hates(e) \wedge hater(e,X) \wedge hatee(e, COMP550) >$
`S -> NP VP`

`NP` $< \lambda P.\lambda Q.\forall x.student(x) \rightarrow \neg Q(x) >$
`NP -> DET N`

`DET` $< \lambda P.\lambda Q.\forall x.P(x) \rightarrow \neg Q(x) >$
`DET -> no`

`N` $< \lambda x.student(x) >$
`N-> student`

`VP` $< \lambda z.\exists e.\ hates(e) \wedge hater(e,z) \wedge hatee(e, COMP550) >$
`VP -> V PN`

`V` $< \lambda w.\lambda z.w[\forall x.\exists e\ hates(e) \wedge hater(e,z) \wedge hatee(e,x)] >$
`V -> hates`

`PN` $< \lambda X.x(COMP550) >$
`PN -> COMP550`

1.3. **c.** : The representation of wants is:

$$\exists e wants(e) \wedge wanter(e, s_1) \wedge wantee(e, s_2) \tag{1.7}$$

One interpretation is to start by looking at s_1, the predicate *there is an exam y that is wanted by* s_1 which then follow by *for each student x, negate this predicate over* s_1:

$$\begin{aligned}&(\lambda Q.\exists y.exam(y) \wedge Q(y))(\lambda s_2.\exists e.wants(e) \wedge wanter(e, s_1) \wedge wantee(e, s_2))\\ &\quad =\exists y.exam(y) \wedge \exists e.want(e) \wedge wanter(e, s_1) \wedge wantee(e, y)\end{aligned} \tag{1.8}$$

(1.9)
$(\lambda Q.\forall x.student(x) \rightarrow \neg Q(x))(\lambda s_1.\exists y.exam(y) \wedge \exists e.want(e) \wedge wanter(e, s_1) \wedge wantee(e, y))$
$=\forall x.student(x) \rightarrow \neg[\lambda s_1.\exists y.exam(y) \wedge \exists e.want(e) \wedge wanter(e, s_1) \wedge wantee(e, y)](x)$
$=\forall x.student(x) \rightarrow \neg\exists y.exam(y) \wedge \exists e.want(e) \wedge wanter(e, x) \wedge wantee(e, y)$

Interpretation 1: There is no exam anywhere that any students want.

A second interpretation follows from starting with s_2: *There is an object s_2 that is not wanted by any student x*, and then set the object to be the exam y, which can have property Q (not being wanted).

(1.10)
$$(\lambda Q.\forall x.student(x) \rightarrow \neg Q(x))(\lambda s_1.\exists e.want(e) \wedge wanter(e, s_1) \wedge wantee(e, s_2))$$
$$= \forall x.student(x) \rightarrow \neg(\exists e.want(e) \wedge wanter(e, x) \wedge wantee(e, s_2))$$

(1.11)
$$(\lambda Q.\exists y.exam(y) \wedge Q(y))(\lambda s_2.\forall x.student(x) \rightarrow \neg(\exists e.want(e) \wedge wanter(e, x) \wedge wantee(e, s_2)))$$
$$= (\exists y.exam(y)) \wedge [\forall x.student(x) \rightarrow \neg(\exists e.want(e) \wedge wanter(e, x) \wedge wantee(e, y))]$$

2. Lesk's Algorithm

2.1. **General Approach.** For all experiments, I worked with a development set of $n = 194$ and a test set of $n = 1450$. Hyperparameter tuning was accomplished using randomized search with (100 trials), picking random values for all hyperparameters between $0 < \lambda < 1$.

2.2. **Baseline Models.** The baseline (the #1 sense as indicated by WordNet) substantially outperformed NLTK's built-in Lesk algorithm on the dev set (50.25% accuracy vs. 26.80% accuracy) and test set (49.38% vs. 25.86%). This is likely because WordNet ranks word senses by their likelihood in real text, so it is reasonable that the most likely sense is a strong baseline. Furthermore, the Lesk algorithm is not well-equipped to deal with limited context, e.g., *'d001.s029.t001: The Heavyweights are coming'*.

2.3. **Simplified Lesk.** To address this, I developed 2 approaches built on top of the simplified Lesk algorithm. Simplified Lesk algorithm [2] uses, for each lemma l_i, the entire sentence minus the lemma as context c_i, instead of a using fixed context window size around the lemma. It also expands the glossary g_i to include all available examples from each considered synset s_i, and if no good candidate s_i is found, defaults to using the #1 sense indicated by WordNet. This approach performs close to baseline (dev set: 45.36%, test set: 43.66%), likely due to the initialization. As is standard, this algorithm selects the sense with maximal overlap between c_i and g_i. No hyperparameter tuning was performed.

2.4. **Distributional Signal: Synset sense frequencies in our corpus.** To incorporate distributional information, I sought to incorporate distributional information regarding the distribution of *synset sense frequencies appearing in our corpus*. To do this, I extracted the lemma counts from all lemmas and all synsets. Then, I normalized these lemma counts by the inverse frequency of the word occurrence in our corpus S (tf-idf [4]). I used Laplace smoothing to deal with 0 counts. Therefore for each sense frequency f_i:

$$f_i = \frac{\#lemma_i + 1}{S_i + |lemma|} \tag{2.1}$$

This distributional signal D was then used to weigh the candidacy of each synset under consideration,the multiplied by a tuning hyperparameter λ_D. To facilitate the mixing of this signal and the simplified Lesk signal L, I introduced a second hyperparameter λ_L. Therefore the total candidacy score C for a synset is calculated as:

$$C = \lambda_L * overlap(c_i, g_i) + \lambda_D + getSynsetFreqs(s_i) \tag{2.2}$$

I found reasonable settings of λ_L and λ_D using randomized search as detailed above.

I tried defining the corpus as all the words in all the definitions of our development set, but this produced poor scores on the test set due to overfitting (recall the dev set is small). As a follow up, I defined the corpus as the entire SemCor corpus, as in previous work [1], which hopefully gives a more general estimate of word frequencies in text.The randomized search found a best dev set accuracy of 38.66%, $\lambda_L = 0.071$, and $\lambda_D = 0.65$. Test set performance with these parameters was 43.38%. It is interesting that this method performs so much better on the test set, likely due to the larger corpus used in learning synset distributions, but disappointing that it under-performs both the baseline and simplified Lesk implementation.

2.5. **Word Embedding Signal: Cosine Distance Between Word2Vec Embeddings.** As a second attempt at incorporating outside distributional signals into our model, I turned to Google's pretrained Word2Vec model [3]. Briefly, I built our c_i, g_i, and corpus S as normal. Then, to compute the overlap between the two, I took the mean word vector across all context and glossary words weighted by their inverse frequency in the corpus f_k:

$$V_c = \frac{1}{|c_i|}\sum_{k=1}^{|c_i|} getEmbedding(c_i^{(k)})f_k \quad V_g = \frac{1}{|g_i|}\sum_{k=1}^{|g_i|} getEmbedding(g_i^{(k)})f_k \tag{2.3}$$

And then computed the cosine distance between these weighted mean vectors. The intention here is to capture information distributional information about word co-occurrence that may occur outside the dev-set distribution to correctly classify senses. As before, I used a hyperparameter λ_W to control the contribution of this embedding signal. Therefore the total candidacy score C for a synset is calculated as:

$$C = \lambda_L * overlap(c_i, g_i) + \lambda_W + embeddingSimilarity(c_i, g_i) \tag{2.4}$$

The hyperparameter search yielded an accuracy of 33.51% on the dev set, with $\lambda_L = 0.060$ and $\lambda_W = 0.72$. Test set accuracy was 32.83%. Both scores far below the baseline and basic lesk. I hypothesize that this method is too powerful for it's own good – taking into account lots of word co-occourances that do not appear in either the dev set or test set, and therefore misleading the lesk algorithm. I suspect building a word embedding directly on the dev set would perform better, although I am skeptical of this as well, because the dev set was very small.

2.6. **Conclusion.** : I failed to outperform the baseline, which made use of the human knowledge about word-sense likelihoods built into WordNet. This points to the correct successful methods for word sense disambiguation requiring more external information regarding language usage patterns their relationships to senses. I suspect a model that uses word sense probabilities, as shown, mixed with word embeddings trained on the dev corpus, to likely perform better than the results presented here. A larger development set would also be invaluable for being able to tune hyperparameters that generalize to the test set well.

References

[1] Pierpaolo Basile, Annalina Caputo, and Giovanni Semeraro. An enhanced lesk word sense disambiguation algorithm through a distributional semantic model. pages 1591–1600, 2014.

[2] Adam Kilgarriff and Joseph Rosenzweig. English senseval: Report and results. 6:2, 2000.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[4] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. 242:133–142, 2003.

McGill University
Email address: joseph@viviano.ca