CS 288: Reranking

Anting Shen
23566738
antingshen@berkeley.edu

1 Introduction

2 Learning Algorithms

use best F1 as gold

3 Features

Using best-list position and rule indicators as simple features, the reranker has 45,991 features, trains in 110 seconds, and achieves a F1 of 85.03.

I then added the 1-width binned log probability assigned to a tree as an indicator feature. While Charniak and Johnson used the probability as a feature directly, Johnson and Ural's used both real probability as well as binned log probability indicators, and the binned indicators had a greater effect on F1. This may be due to binning probabilities allows the reranker to accurately learn nonlinear relationships between log probability and tree quality. This feature increased F1 to 85.21. Adding lengths per rule further increased it to 85.24.

Many potential features involve some sort of lexicalization, which comes with the issue of sparsity where words aren't seen frequently in training data. To combat this, I borrow the idea from the Less Grammar, More Features paper of representing words as their longest suffix that occurs more than 100 times. As a preprocessing step, a suffix trie with counts is constructed from all tree yields, then any nodes with fewer counts than threshold are pruned before feature calculation step. [1]

Using this new technique, I added features to mark the starting and ending words of rules. This captures some information about the lexical and syntactic heads of the trees. Case sensitivity was tested, and it made no difference so case insensitive was chosen for performance. Adding these features brought feature count to 143,529. It trains in 153 seconds and achieves a F1 of 85.63, which is now more than 1.5 points above the 1-best baseline. Additonally adding start and end tags increases the F1 to 85.64.

To learn the context of trees, I fire features for the word before and after each subtree rule. This increases feature count to 217,879, training time to 177 seconds, and F1 to 86.23. Adding neighboring tags decreased F1, but separating the neighboring tag features by subtree length as was done in Charniak and Johnson, the F1 increased to 86.26. Varying the l_1 and l_2 parameters (number of neighbors to fire on for each side) reduces F1, so a value of 1 was used for both.

I then noticed a fair portion of subtrees have large spans, creating features that are sparsely fired when combined with other span qualities. Following the Hall paper, I binned the lengths into 1, 2, 3, 4, 5, 10, 20, 21+, and applied binning to all previous instances where unbinned length was used. This increased F1 to 86.48.

So far these features have been applied only to trees whose rules are not ROOT or S, since S tend to occur infrequently and usually are not the reason parsers make mistakes. Experimenting with firing features on S tags as well reduces F1, as these additional features overfit and explain away relationships better described by other features.

However, for the next feature class, split points, I included S tags since how they split is relevant. Features were fired for the words on either side of a split for binary branching tree nodes to handle a number of cases as described in the Hall paper. Doing so increases number of features to 450,110, training time to 199 seconds, and F1 to 86.71.

Next, I added span shape where each span is described by each word's characteristics: whether it is uppercase, punctuation, or digit. This did not have an effect on F1 when done for all spans of all lengths. When done for only spans of size 5 or smaller, as shape should be most important for short spans with punctuation, F1 remained at

[1] The word "SEAQ" presented itself as an interesting edge case, and was represented as empty string

86.71. I then added back in larger spans, but only took the first and last two words to prevent sparsity, and the F1 decreased, so this feature class was abandoned.