CS 4650 Final Project Report

Andrey Kurenkov, Charles Wang, Stefano Fenu

Implementation - Compulsory

Main.py is reproduced within the ipython notebook (project.ipynb) which is zipped in project_source_RST.zip and allows for running the compulsory code. See the readme included for instructions about executing the rest of the source code. Ablation tests were run on the full feature set (including parts from the independent part of the project) and the results for features marked with * reflect ablated features. Results are summarized below from all attempts to improve RST parsing through additional features:

	F1 Nuclearity	F1 Span	F1 Relation
Out-of-box	.327	.573	.205
+Compulsory	.626	.797	.488
+Same Sentence*	.66	.831	.529
+NER*	.662	.836	.534

*These trials were performed using a classifier trained on the full feature set, whereas the previous two were performed using a classifier trained on the limited feature sets.

The compulsory part was implemented by adding fields to the SpanNode object, adding code to initialize these fields (last POS, first POS, headset, entities, etc.) correctly, and adding code to handle merging these fields across nodes. Most changes were made to the files buildtree.py and parser.py. Most other changes were superficial and served to aid in passing around files or suppressing spurious print output. Stanford's POS tagger, downloaded from (http://nlp.stanford.edu/software/stanford-postagger-full-2014-08-27.zip), and dependency parser, downloaded from http://nlp.stanford.edu/software/stanford-parser-full-2014-08-27.zip, were used to get POS tags and dependency parses. These tools are integrated fileParser.py and the code to process the output is there as well. POS tagging was done exactly by obtaining the POS tags for the first and last token in every EDU. This is easily merged by taking the first POS of the left node and the last POS of the right node. Obtaining the headset was done exactly by first identifying the number of tokens in each span. Then mapping each token to its index in the document. Finally, each word's parent word is labelled by token index. If the parent word's index is outside of the token range of the span in which the word is present, then the word is a headword. This leads to the natural merging operation of expanding the token range, and reevaluating whether a token's parent word is included within that range or not. All other features were simple to extract from data already present.

Independent: Improving RST

Entity recognition and coreference also seem useful for making local decisions. If two adjacent spans mention the same entity, it's likely that we should merge them. Furthermore, the appearance of different entities in nearby spans might indicate a change in topic or otherwise serve to semantically separate these spans. However, it seems that while applying features based on entity recognition and coreference might help to make better local decisions, it does not seem likely that it would help make good long-range decisions simply because the most common entity is likely to dominate and merging too greedily might lead to incorrect parsings.

To get these features, we first run the text through a NER system, from http://nlp.stanford.edu/software/stanford-ner-2014-08-27.zip, to extract entities and a coreference resolver to identify subsequent appearances of an entity. Then, features are made corresponding to the set of entities appearing within a span, whether the same entity appears in nearby spans, and whether different entities appear in nearby spans.

Although the proposal was to find and integrate a coreference system, it's clear that just using features regardless of the source is not a good idea. In particular, as we saw in class, the integration of RST features into other parsing systems seemed to decrease performance because RST parsing was not good enough to be used effectively. We could not find a coreference system that seemed to perform well enough to be effective.

We did, however, incorporate NER features into the data. In particular, we included features for the types of entities mentioned within one span as well as the number of entities that appeared in a span. The last feature actually decreased performance. We suspect this was because as more and more merging happened, eventually one span had many mentions while the other had few and this feature didn't really provide much information about the spans other than that one was longer than the other. However, with the types of named entities, we do see minor increases in performance.

As a fallback,we also implemented a feature for whether two spans were in the same sentence or not (or for merged spans, whether two spans had any sentence overlap). Somewhat surprisingly, this feature provided a large performance benefit. This is surprising because it seems like this information should already be present in some of the other features. In particular, the last word and first word / last POS and first POS of adjacent spans (queuespan1/stackspan1 and stackspan1/stackspan2). For example, if a sentence ending punctuation does not appear at the end of a span, then those two spans should be in the same sentence. The results from this section are included in the table presented above and thus have not been reproduced below.

Independent: Beam Search

Action selection by the provided parser operated in a very greedy fashion, solely reliant on the state of the stack and queue. In an attempt to improve parse selection, we implemented a beam search. Though we initially suggested to compare this to A*. It was expected and later confirmed by another group that full A* is too expensive on this data given its branching factor, and the heuristics needed to power it well required the implementation of the outside viterbi algorithm (see Klein and Manning). Therefore, we decided to focus on testing beam search and extensions of it. During testing it was discovered that using headword features caused the evaluation to be too expensive, leading them to be excised. Beam search was evaluated using the following heuristics:

LatestWeight: Actions on the beam were ranked solely by their prediction scores. In the case where the beam width is 1, this is identical to the original best-first search.

DecayingWeight: All prediction scores for each parser state were maintained, and these were summed, exponentially decaying from the latest result. The idea behind this heuristic is that one mediocre action choice should not completely discount a long string of very good ones. Different rates of decay were used, but none of these variations were found to improve the performance significantly. The best results are shown in the table below.

The idea with the following two heuristics was to incorporate information not used by the classifier. With the above two heuristics we were simply reusing the scores output by the classifier, which seems slightly redundant since that's what the vanilla shift-reduce parser used. Then since the shift-reduce parser never looked at tree features, we tried some heuristics that captured information regarding the structure of the parse tree in addition to using the scores output by the classifier.

TreeBalance: In an attempt to account for the global structure of the resulting parse tree, an expected tree balance was computed for each tree size, based off of labeled trees in the training set. Successive penalties were applied to actions that moved the structure of the tree several deviations away from the expected values.

RelationProportion: Since prediction scores did not reflect the relational structure of the entire tree, this heuristic was an attempt to account for the global structure of the tree. The expected relational proportion was computed off of labeled RST trees in the training set, and successive penalties were applied to the weights if proportions were found to be several deviations away from the expected values.

Results are summarized in the table below. The baseline is the full feature set minus the headword features and otherwise the rows are in pairs of (heuristic, beam width). The defaullt values were DecayingWeight:0.3, TreeBalance:0.05, and RelationProportion:0.005.

Heuristic	F1 Nuclearity	F1 Span	F1 Relation
-headwords	.655	.835	.523
LatestWeight, 1	.655	.835	.523
LatestWeight, 5	.639	.827	.496
LatestWeight, 10	.630	.824	.487
DecayingWeight, 1	.655	.835	.523
DecayingWeight, 5	.649	.830	.511
DecayingWeight, 10	.647	.829	.509
TreeBalance, 1	.655	.835	.523
TreeBalance, 5	.646	.832	.500
TreeBalance, 10	.639	.830	.491
RelationProportion, 1	.655	.835	.523
RelationProportion, 5	.625	.822	.477
RelationProportion, 10	.641	.832	.493

As expected for LatestWeight and DecayingWeight, with beam width 1, we simply reproduced the output from the baseline classifier. This is because the algorithm is just taking the best option at each point, which is equivalent to the original shift-reduce parser. Unexpectedly, perhaps, the tree features also showed the same behavior with a beam width of one. This suggests that looking at the trees doesn't really provide much more information about the correct action to take than just looking at the features already present. This is possibly a side effect of the large amount of structural features already incorporated into the baseline classifier: perhaps there is already enough information about the tree even present in the top two elements on the stack and first element on the queue. Unfortunately, increasing the beam width actually decreased the performance.

Unfortunately, increasing the beam width seemed only to decrease the performance (except from width 5 to 10 for RelationProportion). This could happen since we increase the number of actions to consider at each point. With more actions, we may explore trees which had one relatively bad local decision followed by a lot of attractive options that overrode the baseline. In theory, with infinite beam width, we should always do at least as well as the baseline. However, given the massive branching factor possible for the RST tree, this is not feasible in practice. Another possible fix would be allowing for backtracking, but this is slightly moot since we really

have no intuitive grasp of when to backtrack (what does a bad RST tree look like?).
Independent: Application

Lastly, we also explored using RST features to assist with assessing the helpfulness of movie reviews. The SNAP dataset contains 8 million Amazon movie reviews, with metadata that includes the helpfulness ratings users of the site gave to different reviews. We initially planned to use RST features for sentiment analysis with this dataset, but that has already been significantly researched and in particular multiple successful techniques for using RSTs for improving sentiment analysis has been published. Unlike sentiment analysis, very little work has been done on the subject of helpfulness and it seemed likely that the bag of words representation would not work as well for it since ngrams do not map directly to any helpfulness metric. Furthermore, we thought that RSTs were a good fit for this problem because they offer a way to measure coherence in a text and high coherence would likely make a review helpful and vice versa. Our plan for using RSTs for features was to initially extract as many simple features (such as counts of relations, depth, number of EDUs) as possible from the RST, as it was expected this would reveal the complexity and depth of a review in a way ngrams could not.

A significant amount of time was spent getting initial results with no RST features to later compare against. The outcome of this work can be found in helpfulness/reviewHelpfulnessClassification.py and reviewDatasetParsing.py. The former contains all code pertinent to learning and evaluation classifiers based on different features, and the later provides various generator-based parsing functions that enable handling so much data. Note that the SNAP dataset needs to be in the same directory as the code to run, but due to being 8.6GB is not included in the submission. To simplify the problem, only reviews with at least 6 helpfulness ratings were used and binned into ‘good’, ’okay’, and ‘bad’ reviews based on the ratio of good to overall ratings.

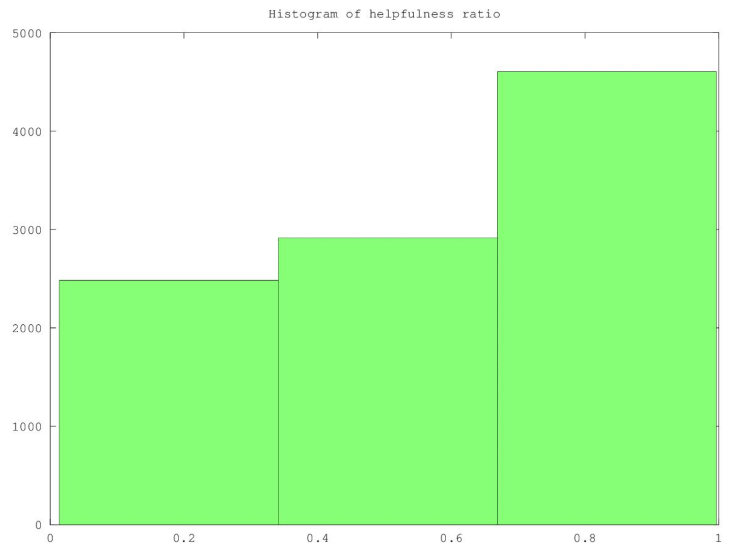

FIgure 1. Distribution of classes in the dataset; roughly %45 percent is one class.

As seen in Figure 2, leveraging the large amounts of data allowed for good performance using nothing but the tfidf statistics for a set of ngrams based on a subset of the data. After evaluating how well the task could be accomplished using only tfidf ngram features, the RST parser was sufficiently worked on to use for feature extraction on the movie dataset. In order to do that, we first integrated the SLSeg discourse segmenter package to extract EDUs from the movie review texts. It is a rule based segmenter that just stores many common phrases or words to segment by. The output of the package required a large amount of post-processing and handling of edge cases in order to work with the existing pipeline for parsing lists of EDUs, which needed to be done before learning over the RST trees of movies reviews could be done.

After that was done, a major impediment we did not expect was that the feature extraction necessary for the RST parser was very time consuming, and also only 11197 reviews could be initially be parsed. Of those, 4736 had at least 3 ratings, and so 3947 instances were used used for training and the rest for test set. Feature were extracted from the text representation of the RST, and included the number of EDUs, number of nuclei and satellites, count of each relation type, and total tree size.

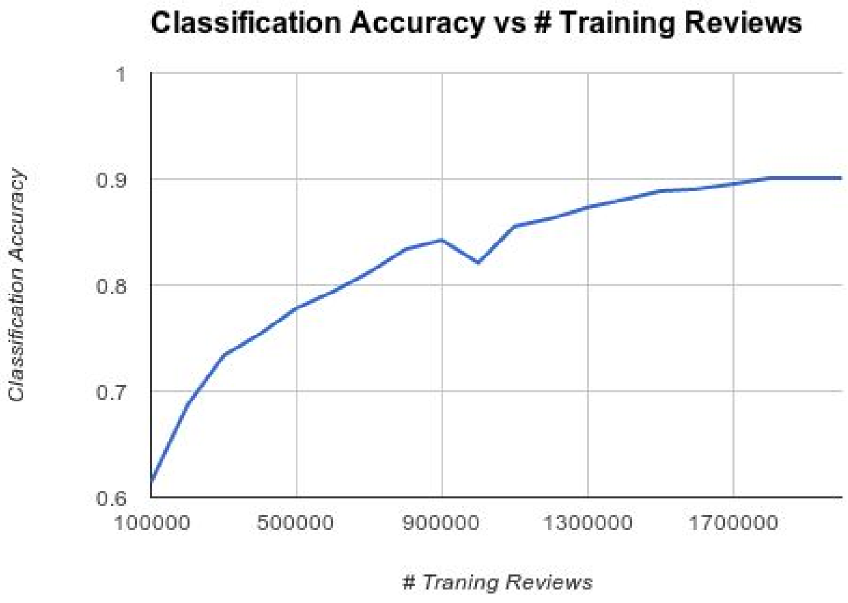

Figure 2. Results for using length 1-3 ngram tfidf features, based on the vocabulary of 50k reviews, evaluated on a set of 50k different reviews using a PassiveAggressiveClassifier

	Random Value Features Baseline	Text Length Baseline	Label Distribution Sampling Baseline
Passive Aggressive	0.2188	0.2188	0.3465
Multinomial NB	0.4508	0.4508	0.3769

Table 3. Baseline results for two different classifiers. Notably, Passive Agressive seems to overfit on bad features whereas MultinomialNB ignores them and get majority performance.

	ngrams	Full RST feature set	Limited RST feature set	Limited RST + ngrams
Passive Aggressive	0.4913	0.3110	0.4366	0.4853
MultinomialNB	0.4245	0.4539	0.4539	0.4518

Table 4. Results with different features. As with other bad features, the full set of RST features made the Passive Aggressive classifier perform worse but did not impact MultinomialNB

The results for tests with this set of data are summarized in Table 4 and Table 5. Using the full set of RST relations as features actually made the Passive Aggressive learner perform worse, which was also the case with other bad features - it appears that the passive aggressive algorithm can overfit on the training set with bad features, whereas MultinomialNB does not. It was decided that inspecting the features of the movie review RSTs to find the best features was a worthwhile task. Several functions, for producing plots of data and correlation coefficients between features and helpfulness, were implemented in reviewDatasetInspection.py. Appendix 1 has full correlation results based on scipy.stats.pearsonr. Limiting the RST feature set to several top ranking features increased performance significantly, though some of the highest ranked features still led to bad results. It may be that none of the features has strong enough correlation to be used well for classification, as they simply overfit to subsets of the data.

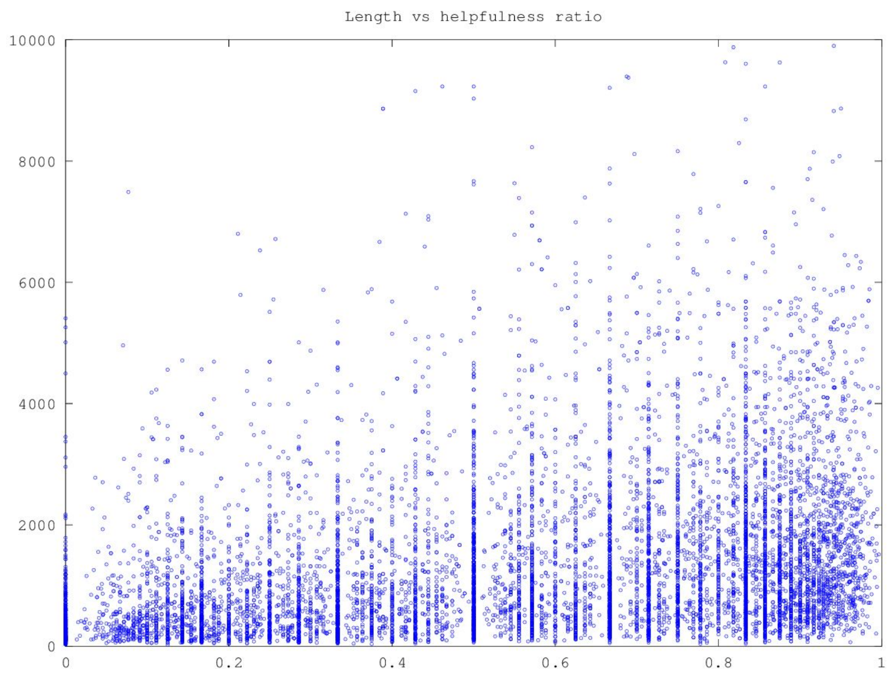

Figure 3. Despite intuitively seeming like a good feature, length does not correlate very strongly with helpfulness. This is one of the results from reviewDatasetInspection.py

The best results that were gotten for the limited RST feature set were through using the ste ['conclusion','proportion','result','antithesis','circumstance','otherwise']. Though it is unexpected that even using RST features that correlate with the label results in a performance decrease, the code for this is thoroughly checked. We therefore had to conclude that the fault is still with the features not being strong enough, so that the PassiveAggressive classifier simply overfit as it does with random features. To overcome this lack of performance gains, we had two solutions: to simply perform more parsing and get more RST data, and to attempt to find better features by implemented further feature extraction over the RST tree (contained in the added TreeStatistics.py file) and find their correlation with the label. The additional feature implemented were tree depth, tree balance, and the addition of parent relations in addition to just relation counts. The best correlated features from this large set of features can be seen in Appendix 2.

Due to several technical problems, the second set of parsed movies reviews we generated only had 3975 reviews with 3 or more helpfulness responses which were similarly split into a train and test set. Results for tests with this second set of data are in Table 5 and 6. Though the correlation check found multiple seemingly good features, there was not a significant gain from RSTs as before. A notable difference is that the Passive Aggressive classifier ignores bad baseline features just like MultinomialNB, which is unexpected given the previous results. Since the code is virtually the same for getting these two results, there is no clear explanation for why this happened. Otherwise, the conclusions are the same - despite having some RST features that displayed correlation to the label, the correlation was too weak to lead to the RST features being particularly good features. It seems that with such a small amount of training data, a more complicated feature such as the RST-based coherence measuring described in Dias et. al would be beneficial. However, implementing the algorithm to do such coherence measuring was beyond the scope of what we could do for this project. Since the RST features do clearly help with training, it is reasonable to expect that given more data the classifiers could better use them and avoid overfitting just as was the case with ngrams.

	Random Value Features Baseline	Text Length Baseline	Label Distribution Sampling Baseline
Passive Aggressive	0.3861	0.3861	0.3496
Multinomial NB	0.3861	0.3861	0.3534

Table 5. Baseline results for two different classifiers. Notably, Passive Agressive seems to overfit on bad features whereas MultinomialNB ignores them and get majority performance.

	ngrams	Full RST feature set	Limited RST feature set	Limited RST + ngrams
Passive Aggressive	0.4528	0.3899	0.3874	0.3861
MultinomialNB	0.3874	0.3496	0.3823	0.3849

Table 6. Results with different features. As with other bad features, the full set of RST features made the Passive Aggressive classifier perform worse but did not impact MultinomialNB

Appendix 1

Correlation values for initial feature set.

Feature (mostly counts)	Correlation	Probability of uncorrelated data getting this correlation
comment	-0.010415	0.464519
interpretation	0.017728	0.213108
sequence	0.012274	0.388721
lines	**-0.026215**	**0.065587**
antithesis	0.01485	0.297005

enablement	0.002268	0.873431	
evidence	-0.013395	0.34686	
topic	-0.021402	0.13281	
manner	-0.015498	0.276419	
Satellite	-0.022583	0.112718	
disjunction	-0.017398	0.221758	
span	-0.022583	0.112718	
purpose	-0.001438	0.919541	
means	-0.008323	0.558892	
question	0.000982	0.944993	
same	-0.011718	0.410536	
list	-0.018937	0.183532	
Nucleus	**-0.027186**	**0.056205**	
statement	-0.009789	0.491824	
rhetorical	0.004012	0.778137	
cause	0.006006	0.673179	
contrast	-0.003636	0.798465	
conclusion	**-0.024532**	**0.084883**	
elaboration	-0.020545	0.149051	
attribution	-0.001766	0.901316	
analogy	0.003377	0.812516	
circumstance	**-0.030865**	**0.030161**	**Did not help**
explanation	**-0.0264**	**0.063699**	
problem	-0.002605	0.854837	
reason	-0.008233	0.563142	

maxDif	**-0.026518**	**0.06253**	
background	0.01577	0.268076	
inverted	-0.007546	0.596151	
contingency	-0.013303	0.350167	
condition	-0.002398	0.86627	
comparison	-0.001956	0.890773	
temporal	-0.011795	0.407489	
textualorganization	**-0.029492**	**0.038312**	**Did not help**
evaluation	-0.004656	0.743697	
maxEDU	**-0.026377**	**0.063931**	
preference	-0.00333	0.815126	
summary	**-0.025913**	**0.06875**	
definition	-0.016233	0.254271	
hypothetical	0.020931	0.141552	
restatement	0.009874	0.488036	
proportion	-0.021713	0.12727	
consequence	0.002546	0.858095	
concession	-0.002645	0.85262	
otherwise	**0.02683**	**0.059504**	
example	-0.01633	0.251451	
result	-0.016389	0.24973	

Appendix 2
Correlation values for best results in second feature set.

Feature	Correlation	Probability of

			uncorrelated data getting correlation
('statement'	'condition')	0.062272	0.000085
('comment'	'concession')	0.047295	0.002848
list	-0.033689	0.033609	
('cause'	'cause')	0.0489	0.002035
('result'	'explanation')	0.067721	0.000019
('contrast'	'contrast')	0.037712	0.017377
('list'	'explanation')	-0.032778	0.038711
('concession'	'purpose')	0.035219	0.026332
('contrast'	'comment')	0.045266	0.004296
('summary'	'means')	0.067721	0.000019
('condition'	'circumstance')	0.036579	0.021046
('comparison'	'antithesis')	0.03304	0.037178
('textualorganization'	'antithesis')	0.035981	0.023243
circumstance	-0.04482	0.004693	
('manner'	'means')	0.049295	0.001871
('consequence'	'antithesis')	0.062272	0.000085
size	-0.038667	0.014731	
statement	0.039046	0.013784	
elaboration	-0.038872	0.014211	
explanation	-0.032465	0.04061	
('condition'	'textualorganization')	0.031768	0.045119
('elaboration'	'span')	-0.038708	0.014627
('purpose'	'preference')	0.054362	0.000603

('means'	'contrast')	0.0354	0.025566
('statement'	'span')	0.04607	0.003657
('contrast'	'same')	0.035347	0.025789
('consequence'	'purpose')	0.052162	0.000998
('elaboration'	'statement')	0.068413	0.000016
('explanation'	'result')	0.039306	0.013165
('question'	'contrast')	0.0709	0.000008
('antithesis'	'reason')	0.039419	0.012905
same	-0.031263	0.04865	
('purpose'	'disjunction')	0.043509	0.006059
('statement'	'elaboration')	0.046451	0.003386
('span'	'same')	-0.035319	0.025905
('span'	'span')	-0.036758	0.020426
sequence	-0.033437	0.034958	
('elaboration'	'elaboration')	-0.038227	0.015902
('reason'	'textualorganization')	0.036318	0.021983
('comment'	'means')	0.078953	0.000001
('reason'	'contrast')	0.037471	0.018107
('antithesis'	'cause')	0.062296	0.000084
span	-0.039593	0.012512	
('span'	'circumstance')	-0.040722	0.010209
('summary'	'restatement')	0.05201	0.001032