NLG for Fantasy Sports

John Maxwell
University of Illinois at Chicago
jmaxwe3@uic.edu

Vaibhav Singh
University of Illinois at Chicago
vsingh38@uic.edu

Abstract

In this work we present a system for hierarchical natural language generation in the domain of fantasy sports news. We automatically generate templates from a source corpus and use data-to-text strategies to generate summaries of a player's performance in a game.

1 Introduction

In fantasy sports users draft their own team of players to compete each week and get points reflecting their teams' performance in real life games. News updates about the performance of players enhances the experience for users, but with many players in numerous games each week across different sports this is a lot of content to create on a regular basis. For this project we automated the generation of news about players in the domain of fantasy football.

We follow the work of Reiter and Dale (2000) to inform many of our approaches to thinking about this problem. A common approach to bootstrap content creation is to create templates from an existing source of text in the desired domain. We gather a parallel corpus of data and text about a particular player and game to generate these templates. We then generate text hierarchically by performing document planning, microplanning, and surface realization.

Templates improve microplanning and surface realization by maintaining syntactic and semantic structure from the source corpus. Where text aligns with data in our table, we can create slots in the templates for inserting future data and generating new text. The downside of templates is that they lack variation, so we break the templates from the source text into chunks which we can recombine creating a balance between these aspects. We briefly describe the hierarchical steps here and elaborate further in section 4. Document planning involves choosing what type of information we want to talk about. Then microplanning is about how to say it by choosing a template chunk of that type. Finally, we insert the table data into the corresponding template slots to create the surface realization.

Our goal is that the generated text is informed by the data and meets certain requirements of data-driven journalism as described by Leppänen et al. (2017). The desired text should be factually correct and present the most relevant facts; fluent text free of grammatical errors; and be enjoyable to read.

2 Related Work

Weather and sports (Barzilay and Lapata, 2005; Reiter et al., 2005; Chen and Mooney, 2008) have been popular domains for NLG because they contain a lot of numerical data and less syntactic variation than other domains. Much of the prior work in the sports domain has involved generating game summaries whereas we attempt to generate player-based summaries. Our corpus is predominantly one sentence long. Chisholm et al. (2017) similarly focus on generating one sentence biographies from Wikidata.

Wiseman et al. (2017) follow the work of Barzilay and Lapata to create a large-scale corpus of game summaries in the domain of basketball. They conclude that current neural models for data-to-text generation are promising, but currently cannot match the performance of template models. Neural approaches tend to hallucinate facts whereas templates are very good at putting the correct information into the generated text. They also describe how current performance measures such as BLEU are insufficient for analyzing performance the semantic content. As such we rely on other methods of evaluation.

A recent survey by Gatt and Krahmer (2018) contains in depth discussions of many more approaches and problems in NLG.

3 Dataset

We scrapped rotoworld.com for news updates about players and footballdb.com for player statistics. This allows us to construct a parallel corpus for learning data-to-text generation. The news updates were filtered to only summaries of a player's performance in games by only keeping records which have a date matching a game date for that player. We also filter out news updates with keywords corresponding to injuries, contract negotiations, and other non-game related news as that is outside the scope of our project.

We annotated one week of news updates as our gold corpus to compare our approaches against. We ended up with 2,118 training samples and 160 labeled test samples.

4 Methods

For the document planning, microplanning, and surface realization tasks we create features following guidelines in Angeli et al. (2010) section 3.

4.1 Template Generation

We want to generate templates directly from the news updates and parallel statistics data. There are two types of tokens we need to match: numeric data and named entities. For matching named entities, we create glossary of team names with their various surface forms and player names, both of which are known a priori. Surface forms could include the team's city, mascot, city with mascot, abbreviation, or nickname. Surface forms of players could include the player's full name, first name, last name, or nickname. We also normalize number words in the text to their numeric form. With the glossary of named entities and numeric statistic data we can align the text based on exact matches, but conflicts can occur when two fields in the statistics have the same value by chance which happened in 13% of template tags. We need to construct a model to disambiguate these tags.

We train our model on the partially aligned dataset with tags that unambiguously aligned based on the assumptions that: value conflicts occur by chance, so some tags are not more prone to conflicts and there are sufficient examples of each tag being unambiguously aligned because of the previous assumption.

We created features based on the local context around each unambiguous tag: bigrams to the left and right of the ambiguous slot; closest tag to the left and right. Count vectorizers of the context bigrams and context tags are fed into a Naive Bayes model to calculate argmax Pr(tag | context).

4.2 Document Planning

We incorporate domain knowledge into our system by mapping each template tag in our database to a tag type. In document planning we decide "what" we want to talk about by selecting one of these tag types. Table X shows our mapping between tags and tag types.

We use a simple heuristic to chunk the sentences in our corpus into template chunks for document planning. Sentences are split on verbs, prepositions, and postpositions which gives us a sequence of chunks. We rejoin chunks if they contain the same tag type or the latter chunk contains no template tags. This gives us chunks that discuss only one aspect of the player's performance like the concept in Leppänen et al. (2017) that modular chunks correspond to the fact level. Our approach to chunking will not work on many texts, but it is effective for our domain. Our corpus contains single sentence summaries where each tag is talked about only once, and sentence are in a journalistic and consistent style.

The chunking process leaves us with a sequence of tag type chunk such as, *<START><player><receiving><game><END>* . Our goal is to learn the next chunk based on our data and previously generated chunks. At generation time we pick the most likely next chunk base on a discriminative model until we generate the *<END>* chunk or 10 chunks have been generated. In our training and test set we never had to terminate because of the chunk limit. The tag type chunks in our system correspond to the idea of a record in Angeli et al. (2010). The features are: the last chunk type, all previous chunk types, numeric values, and one-hot-encodings of categorical values from the statistics. We exclude the values for *player_name, date, team, opp,* and *away_game* as we assume they have no influence over the final template and in the case of *player_name* and *date* are open vocabularies with

Model	Avg Levenshtein Distance
Exact match only	1.35
Exact match and random choice	0.90
Naive Bayes with context bigrams	0.34

Table 1: Template tagging results (test set)

very large encodings. We use a linear regression to predict the next chunk type.

4.3 Microplanning

Once we have the sequence of chunks types of our document, we want to choose a template chunk of that type to generate in order. We have a set of template chunks from the document planning step which we then filter to chunks that appear at least 3 times in the training set. This removes template chunks resulting from errors in the POS tagging, errors in template tagging, and extreme outlier events. This also improves our ability to learn with statistical machine learning approaches.

Our features for selecting the best template chunk are an encoding of the template tags of the given chunk type, numeric values, and one-hot-encodings of categorical values from the statistics. We once again exclude the values for *player_name, date, team, opp,* and *away_game* with the same justification as described in the document planning section. We use a k-NN with k = 15 and weighted by distance to estimate the probability of selecting each template. We mask the output prediction probability vector with an indicator vector corresponding to the current chunk type to restrict our output. We must predict against all possible templates because each chunk type has different numbers of template choices. We'd either need to create a separate model for each chunk type or learn a binary function to run against each template. Our approach generalizes more than the first and is computationally more efficient than the second. Template chunks are generated for each chunk type.

4.4 Surface Realization

The advantage of templates is that they greatly simplify surface realization. We populate the template tags with their corresponding data values. If the data is numeric, we change any round decimal values to integers. For the *team* and *opp* tags we choose the team mascot surface form if the preceding token is "the" otherwise we choose the team city surface for to make the sentence grammatical.

Team mascot example and context: "... *loss to the Dolphins*". Team city example and context: "... *loss to Miami*".

5 Initial Research

In order to gain insights into existing tools that attempt to solve similar NLG problems, and analyze the contemporary approaches they take, we explored a number of open-source and commercial tools. We have included some of the notable findings, and why they did not meet our requirements.

SimpleNLG (https://github.com/simplenlg/simplenlg). This is an open-source Java API that accepts words, POS forms, and surface-realization configurations to generate semantically valid sentences. An example. The limitations with this tool however, are: (1) Requires configuration of all words and their desired surface forms, (2) does not generate variations of same sentence, (3) does not make use of statistical approach, (4) does not assist with content selection and surface realizations, which were our primary requirements.

Markov chain text generator (https://github.com/barrucadu/markov) is an automatic text generator based on the Markov model. It trains and generates a token depending on the previous token. We set up and experimented with this tool however, found that although the generated text showed variations, they were semantically incoherent. Some limitations we noted with this tool for our purposes were, (1) needs a large corpus for training – our news corpus of around 2 – 3k sentences was not enough, (2) generated text was semantically wanting.

The next solution we explored was **Paraphraser** (https://github.com/vsuthichai/paraphraser). This tool is purposed to paraphrasing sentences when trained on a corpus from the same domain. And as this looked close to what we were looking for – generating novel variations in our news blurbs,

we decided to take a deeper dive into it. Paraphraser is a pretrained bidirectional LSTM encoder and LSTM decoder with attention trained using TensorFlow. It generates variations to text similar to *Quillbot*, a commercial application for generating paraphrases.
After setting up and experimenting with Paraphrases, we discovered the following limitations, however. (1) Pretrained on corpuses that may not be related to the domain of our project, (2) did not yield semantically valid results when tested using phrases from our corpus. We are including an example here.

Source sentence	Paraphrase 1	Paraphrase 2
McCoy has now been held below 10 touches in 2-of-3 games and has handled the ball only 29 times all season.	mccoy 's got a new number of 15 times in the game , and it was still 36 times the whole season .	mccoy is now in 10 minutes of 2-of-3 games , and he has taken care of the ball only 29 times .

Table 2: Paraphraser output

As evident from the output above, the paraphrasing is not semantically precise, which was one of our penultimate requirements – generating grammatically and semantically coherent texts, that are enjoyable to read. Still, analyzing the Paraphraser's open-source implementation paved to the way to our further research into exploring the potential of Deep Learning models for our project, which we share in the following section.

6 Early approaches

After realizing the novelty in our problem and the lack of an existing solution that solves it, we considered several approaches, some of which are as follows.

6.1 Chunking and Combinatorics

We decided to break the news articles into chunks, first, using prepositions, and later using record type.
The former approach to chunking is straight-forward. We split the sentences using prepositions to form semantically valid chunks.

Chunked text	Chunks annotated using fact tags
'Tavon Austin caught 4 passes', 'for 20 yards and a touchdown', 'in week 3', 'against the Sox'.	${player_name} caught ${receptions} passes, for ${rec_yards} yards and a touchdown, in week ${week}, against the ${opp}.

Table 3: Chunked outputs

We then used combinatorics to generate ordered permutations of chunks to then calculate the coherency score of each. The combinations with k-highest scores would then be selected and the tags parsed using the fact database, to generate new news blurbs. Two advantages of this approach were, (1) coherent splitting of semantically valid, atomic chunks that eventually contributed to our final solutions, (2) surfaced the idea of chunk sequencing that led to the next exploration using 'Frequent tag-sets'.

Taking this approach was not selected due to these limitations: (1) Does not make use of a statistical approach to solve the problem, (2) impractically large number of combinations needed to generate variations, yet maintaining semantic validity, (3) the 'Oracle', i.e. coherency score generator is not available. We tried to using a Python language check tool (https://pypi.org/project/language-check/) however, soon discovered that it was capable of detecting only grammatical errors and could not invalidate semantic anomalies or identify valid surface realization in chunk combinations.

6.2 Frequent Tag-Sets using MSapriori

Following from combinatorics, the idea behind this method was to use a statistical approach to generate sequence of tags that were found frequently in sequence, in our annotated chunks. Generating frequent itemset is a concept taken from Data Mining, which helps identify the most frequent items purchased together in a given sequence of shopping basket transactions, per buyer.
To leverage the frequent-itemset generation in solving our problem, we extracted tags from chunk annotations, as explained above, and created a sequence of transactions, similar to the shopping basket sequences of purchased items. Each item in a transaction represented the set of tags in the order they appeared in the respective chunks.

Table 4: MSapriori – Inverse Chunk Dictionary and Combinatorics

Tag-set	Inverse-chunk dictionary			Sample combination
955 : ['${rec_yards}', '${rec_targets}', '${opp}']	${rec_yards}	${opp}	${rec_targets}	for ${rec_yards} yards and a touchdown of ${rec_targets} targets against the ${opp}.
	for ${rec_yards} yards and a touchdown for a ${rec_yards} yard gain while adding a pair ...	against the ${opp}. to the ${opp}. ...	of ${rec_targets} targets of ${rec_targets} passes ...	

We implemented a slightly customized version of the *MSapriori* algorithm to then generate frequent tag-sets, using the above transaction data. Multiple frequent-set samples were analyzed using different minimum supports. Frequent 3 to 5 tag-sets were considered to sequence the respective chunks, using an inverse tag-chunk dictionary, in an attempt to generate semantically valid chunk sequences.

As can be seen in Table 4 above that although it is possible to generate coherently valid sentences, still needs work for surface realization. Some advantages of this approach were, (1) follows statistical approach using minimum support for generating frequent sets, namely *minSup = n(X U Y) / n(S)* where X and Y are tag sequences and n(S) is the total number of transactions, (2) possible to generate valid chunk sequences, but not there yet.

Some limitations that prevented us from choosing this approach were: (1) start tags necessary for initial chunks, e.g. ${player_name} may not be in the frequent sets, (2) the order of tags in frequent sets may not be maintained, (3) large number of chunks in inverse tag-chunk dictionary makes it hard to select valid sequences, (4) text generation will depend on tag-sets, instead of the fact database, which was not in line with our requirements, (5) missing tags in the frequent sets makes surface realization difficult.

6.3 Recurrent Neural Network

Unlike traditional Neural Networks, RNNs are capable of capturing sequences in information, i.e. what came before the current input. They do this using a feedback loop from the output layers to previous, hidden layers.

In exploring this approach, we trained a two-layer LSTM with dense connections using variable sequence length and tested the results. While the RNN was able to learn sentence structures and generated valid and varying text sequences, when trained using a large corpus (eg. Harry Potter), it frequently overfitted on our small corpus of news blurbs.

For the training features and labels, we used one-hot encoding of token sequences, and one-hot encoding of input+1 sequence, respectively. The astute reader would note that the label includes incremental tokens to capture word sequences.

Here we present some sample results using character, chunks, and unigram-based trainings:

Some advantages using RNNs were, (1) text-generation appeared semantically valid even when trained using the small news corpus of 2000 sentences, (2) general model that can be applied cross-domain, (3) unigram training with validation set performed much better as compared to previous approaches.

Observing that the RNN was able to generate valid texts, we were tempted to use it as our final solution, however, the following limitations were thwarting: (1) the model tends to overfit when trained for a large number of epochs. Even though the text was coherent, there were not

Character (epoch=20, loss=0.22)	Chunk (epoch=540, loss=0.719):	Unigram (epoch=139, loss=0.99)
X} ', 'weile adding ${receptions} of ${rec_targets} targets for ${rec_yards} yards ', "in the ${team}' week ${week} loss to the ${opp}."	'${player_name} ' 'caught ${receptions} of ${rec_targets} targets for ${rec_yards} yards and a touchdown ' in the ${team}' week ${week} win over the ${opp}.	pulled through ${pass_attempts} ${pass_yards} yards, and ${pass_td} the week in the ${team}' loss the completed ${pass_completions} of for a yards yards ${game_dow} ${game_dow} ${week} win ${opp}.

Unigram training with Validation Set (train acc=91.5, test acc=80.5)
[" ${player_name} rushed a ${rec_yards} yard in the ${team}' week ${week} ${player_name} ${player_name} a touchdowns in the ${team}' week ${week} win the the ${team}' week ${week} win over the ${opp}", ' ${player_name} rushed ${rush_attempts} times for ${rush_yards} yards and in the of week ${week} against the ${opp}', " ${player_name} caught ${receptions} of ${rec_targets} targets for ${rec_yards} yards and a touchdown in the ${team}' week ${week} loss to the ${opp}"...]

Table 5: RNN outputs

enough variations when we compared the output to our original corpus – a sign of overfitting. (2) With early-stopping of training when the error started increasing on the test set, to prevent overfitting, the text generation was not as coherent. (3) Does not make use of the fact database to generate texts.

Still we saw a lot of potential in how RNN was able to learn even subtle nuances in the text such as capitalization, special tag characters, surface forms, and sequences in general. We would like to explore this approach further for future work relating to NLG and NLP.

7 Results and Discussion

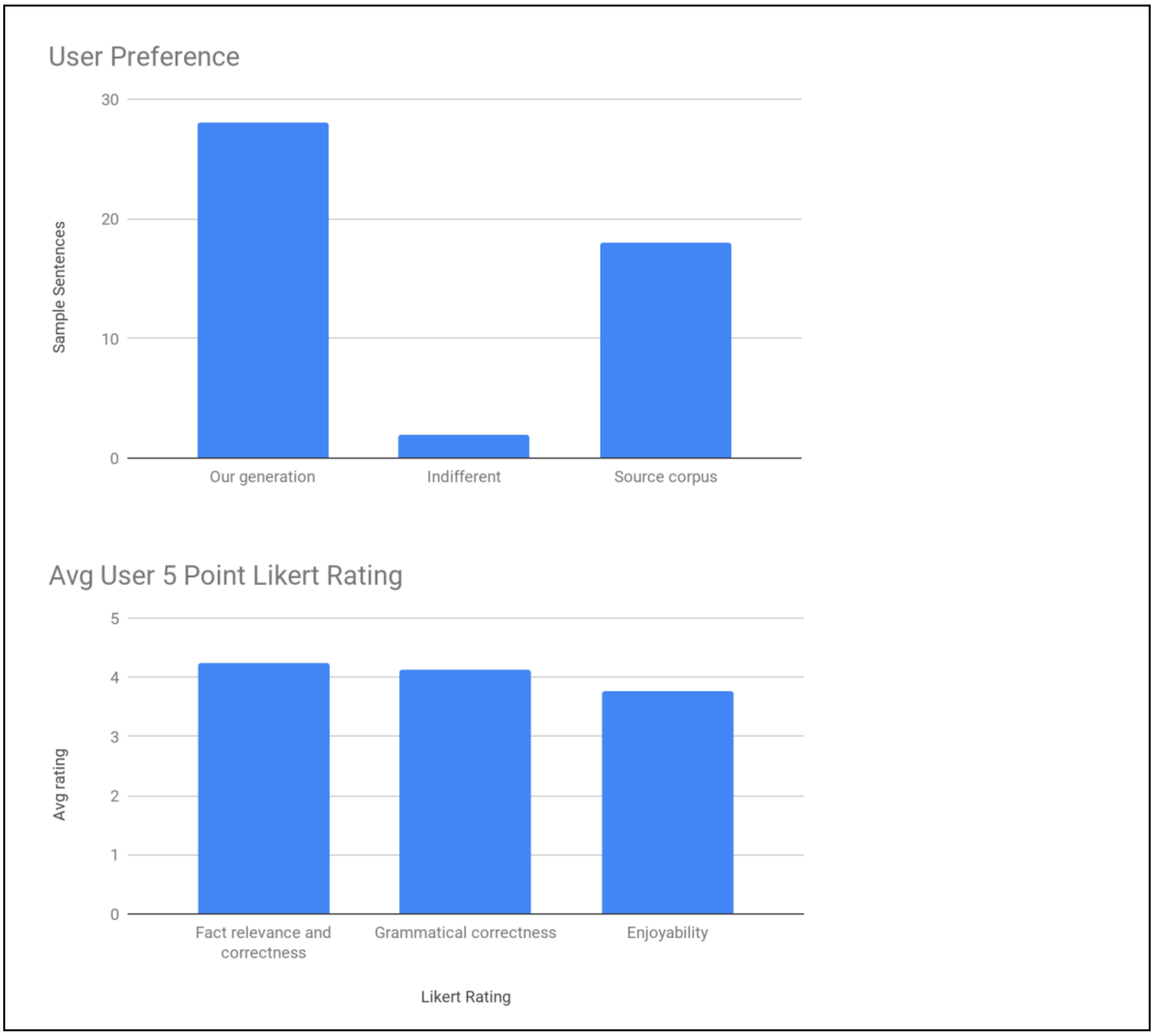
In template generation we use Levenshtein distance to measure the difference between our tagged sequence and the gold standard (Kondadadi et al., 2013). The sequence is represented by the template tags and each chunk of non-tagged text. This allows us to measure 3 criteria of our template tagging: we have the correct tags, the tags are in the correct order, and we're tagging the correct tokens in the text. We compare our method against a baseline of choosing randomly between tags when there are multiple exact matches and find that we can make significant improvements.

All models where chosen with 5-fold cross validation on the training set. Result metrics are reported on the test set. The linear regression model for document planning achieves an average Levenshtein distance of 0.23. The k-NN model for microplanning achieves an accuracy of 0.85 and F1 of 0.76.

Human evaluations are still the best metric for assessing semantic correctness and overall quality of generated text. We structured a two-part survey with 16 respondents and 48 samples from our test set. Each respondent was asked to compare 3 sentence pairs, one from our source corpus and the other generated by our system for the same player-game record. They indicate whether they prefer sentence 1, sentence 2, or are indifferent. If our generated text is comparable quality to the source corpus respondents should not consistently rank the source corpus sentences as better.

We also provide respondents with 3 pairs of generated text and source statistics from the same player-game record. For each pair we ask them to rate the generated text on a 5-point Likert scale for factual accuracy, grammatical correctness, and reading enjoyability. The comparative and

Table 6: User Evaluations



qualitative samples were chosen at random from the test set.

8 Conclusions and Further Work

Although our results look promising there's still a lot of room for improvement according to our three desired generation attributes: factual correctness and relevance; grammatical correctness and fluent text; and enjoyability. In regard to factual correctness templates simplified the process significantly but there are still characteristics that we were unable to match to our statistics data. For example, '... *failed to catch both his targets…*'. Our methods cannot accurately tag the template slot because they aren't represented as numbers of named entities and some logic is required to deduce that *receptions=0* and *targets=2*. A better lexicon could allow us to understand that '*both*' is equivalent to two. Using some sort of first order logic could also help us comprehend the difficult semantic meaning. An example that's common in our dataset is '... *threw for {pass_yards} yards and a touchdown*'. We don't tag '*a*' as one because that could lead to a lot of false positives. Instead we'd need to build a classifier to decide when a non-number word is representing a number. Our microplanning model was unable to learn that this template is only used when *touchdowns=1*.

More data with varied templates may help our models learn mappings like this, but statistical models are generally biased towards more frequent events or templates which limits our variance at generation time. We'd like to explore more deep learning methods like Puduppully et al. (2018) which are able to integrate content selection and planning into an end-to-end neural model.

References

Gabor Angeli and Percy Liang and Dan Klein. 2010. A Simple Domain-Independent Probabilistic Approach to Generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural*

Language Processing (EMNLP '10): 502-512. Association for Computational Linguistics, Stroudsburg, PA, USA.

Regina Barzilay and Mirella Lapata. 2005. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (HLT '05): 331-338. Association for Computational Linguistics, Stroudsburg, PA, USA. DOI: https://doi.org/10.3115/1220575.1220617.

David L. Chen and Raymond J. Mooney. 2008. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning* (ICML '08): 128-135. ACM, New York, NY, USA. http://dx.doi.org/10.1145/1390156.1390173.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from Wikidata. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 633–642. Association for Computational Linguistics.

Albert Gatt and Emiel Krahmer. 2017. Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*. 61. 10.1613/jair.5714.

Ravi Kondadadi, Blake Howald, and Frank Schilder. 2013. A Statistical NLG Framework for Aggregated Planning and Realization. *ACL 2013 - 51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*.

Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. 2017. Data-Driven News Generation for Automated Journalism." In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 188-197.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2018. Data-to-Text Generation with Content Selection and Planning. *arXiv preprint arXiv:1809.00582*.

Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. 2005. Choosing words in computer-generated weather forecasts. *Artificial Intelligence 167, no. 1-2*, pages 137-169.

Ehud Reiter, and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

Benjamin Snyder and Regina Barzilay. 2007. Database-Text Alignment via Structured Multilabel Classification. In *IJCAI*, pages 1713-1718.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*.

A Appendices

Work distribution

Both:

- Background research
- Writing report

John Maxwell:

- Scraped data and news content
- Cleaned and merge data/news to create source corpus
- Player/team glossary
- Coded document planning, microplanning, and surface realization pipeline
- Annotated gold/test corpus

Vaibhav Singh

- Initial research and exploration
- Set up Python environments and tested existing solutions
- Processed data and generated chunks
- Implemented MSapriori, Chunk combinatorics, and Recurrent Neural Network
- Helped in survey planning, survey processing, automating user-feedback evaluation, and reporting