Git Repo: https://github.com/katrina906/CS6120-Summarization-Project

Auto-Summarization for Content Management

Katrina Truebenbach

1. Introduction

Content Management is an essential and yet time-consuming and complex process for any company with sizable Intellectual Property (IP). I work for a management consulting company that creates hundreds or thousands of documents every year that must be assigned appropriate meta-data so that they are findable by an internal enterprise search platform. However, the content creators are often too busy to dedicate time to creating meta-data, especially summaries which take more work than assigning tags etcetera. Thus, this report explores methods for auto-summarizing content and makes a recommendation in the specific context of a management consulting firm.

This auto-summarization tool does not need to create a fully finished, polished summary. Creators should still review, edit, and accept the predicted summary to ensure high quality. Thus, the primary goals are: (1) capture the main themes of the content, (2) preferentially provide too much information over too little information, (3) but do not provide so much information that the task of editing down the summary is equivalent to producing a summary from scratch. In other words, recall is more important than precision, as long as a reasonable measure of precision is retained.

Auto-summarization is a rich field with a large body of existing work. This report uses the specific goals above to evaluate a variety of algorithms that span both extractive and abstractive techniques. Extractive summaries are formed from existing article sentences, while abstractive summaries are generated from scratch. The algorithms will be compared with both quantitative and qualitative reviews that reveal the benefits and limitations of each algorithm. Given this analysis, we ultimately recommend a model that satisfies the three stated goals.

2. Data

The actual content produced by management consulting firms is confidential, so we approximate it with a sample of 10,000 news articles[1] from a set of 90,000 CNN articles.[2] Each article is appended with three "highlights" that can be concatenated to form a gold standard, human labeled summary.

These articles are a good, albeit imperfect, approximation of the target consulting content. The average article is relatively short with 15 to 20 sentences and around 400 words, but there is also a long tail of longer documents (Figure 1). The articles also cover a diverse set of topics, mirroring the many areas a firm may consult in. Finally, news articles often follow the "Pyramid Principle" of writing that is also often used in consulting pitches and case decks in which the main points are summarized initially, followed by more details and evidence. Given this Pyramid structure, we could attempt to use the first several sentences of each article as the summary. This may work for news articles, but will likely be insufficient for the more complex, diverse consulting corpus. We will use this simple idea as a baseline for comparison in Section 4.2, but we will focus on more robust models that consider the entire content when generating summaries.

---

[1] 90,000 articles proved computationally intractable for some algorithms. Instead, a random sample of 10,000 articles is used to train all algorithms. Random sampling avoids including any information that is encoded by the ordering of articles.

[2] The CNN dataset can be found here: https://cs.nyu.edu/~kcho/DMQA//

Before subsampling the full set of articles, we perform some data cleaning. We strip off news source and location information from the article text (i.e., "(CNN) New York …"), drop articles with no text or no summary, and drop articles in which the summary has more words or more sentences than the article text. If a summary has more words than the article, the main article is generally a video or photo with a brief caption, and if a summary has more sentences than the article, the main article is generally not formatted in sentences – for example, a list of soccer game scores. These are not useful articles to summarize.

The gold standard summaries are not used as labels in training because all of the techniques explored in this report are unsupervised. Instead, they are used to both evaluate hyperparameter choices within algorithms and compare the results of different algorithms.

2.1 Length Heuristics

Beyond acting as labels, the gold standard summaries are important to answering the question: how long should the generated summaries be? All the algorithms in this report require us to specify either an exact or a target length.

The first heuristic, as depicted in Figure 2, is that for every 8 article sentences, an average summary has 1 sentence. The second is that for every 12 article words, an average summary has 1 word. Article sentences are on average twice as long as the gold standard summary sentences (Figure 3), so the words heuristic may be more accurate in generating summaries of a similar length to the gold standard. For extractive summarization, in which summary sentences are selected from the existing article sentences, the sentences heuristic will likely result in longer predicted summaries because the heuristic number is based on the shorter gold standard sentences. We will experiment with these two heuristics as hyperparameters and observe how they affect model performance.

2.2 Novel Words in Gold Standard Summaries

A final pre-modeling use for the gold standard summaries is to form some hypotheses around if extractive or abstractive summarization will better approximate the gold standards. Figure 4 shows that on average 80% of the words in a gold standard summary also appear in the article text. This implies that extractive summarization, which only uses words in the article, can approximate the summary well. However, this also shows that the humans generating the summaries also use some new language, implying some use for abstractive summarization as well.

3. Models and Experiments

This report examines several algorithms from two broad types of auto-summarization techniques. Extractive summarization scores all sentences in the article using some algorithm and selects the highest scoring sentences to form the summary. These summaries are guaranteed to be grammatically correct and contain correct information about the article. However, the order in which sentences are selected can be illogical and the entire article sentence must be used, even if it makes one important point followed by unnecessary supporting details, which can result in too-long, too-specific summaries.

Abstractive summarization generates entirely new sentences to form the summary. A common technique is an Encoder-Decoder model in which the article text is inputted as initial context from which novel summary sentences are generated. Abstractive summaries are able to combine concepts from multiple article sentences into one shorter summary sentence. It also does take the order of sentences into account. However, abstractive summaries may not be fully grammatically correct, can generate information not originally in the article, are more computationally intensive to generate, and are harder

to explain to a non-technical audience. This last point is important – if we are creating a tool for content creators to use, they may not trust the summaries generated by a "black box" Neural Network. Meanwhile, it is easy to explain how extractive summarization is simply selecting the "best" sentences from an article.

The next two sections describe the specific extractive and abstractive algorithms used. Then Section 3.3 describes the training process including how we select the best set of hyperparameters or "configuration" for each model.

3.1 Extractive Summarization

We consider three different extractive techniques. The first is a simple method using Term Frequency – Inverse Document Frequency (TF-IDF). The idea is to select sentences that best describe the specifics of the article. First, we train TF-IDF on the entire corpus of articles and get a score for each word in each article. Words with high scores are frequent in the article, but infrequent in the overall corpus. To score each sentence, we average the TF-IDF scores of all words in each sentence and sort the sentences on these averaged scores.

The second technique is TextRank, which is equivalent to PageRank where the vertices of the graph are the sentences in an article and the edge weights are the similarities between pairs of vector representations of sentences. The vertices/sentences that have the highest score once the algorithm has converged are the most representative of the article. Several different vector representations techniques and similarity metrics are considered in Section 3.3.

The final technique is Latent Semantic Allocation (LSA). The goal is to select sentences that capture the most important concepts in an article. LSA uses Singular Value Decomposition (SVD) to decompose, for each article, an input matrix of sentences by vector representations of words in the sentences into several matrices, one of which is a matrix of sentences by vectors of latent concepts in the article. The latent concepts are in order of importance to the article, as represented by the input matrices' singular values/eigenvalues. Each value (i,j) in the matrix represents the importance of the sentence i to concept j. Ozoy et al.[3] describes several ways to select the summary sentences from this matrix. For our purposes, we select the highest scored sentence from the most important concept, then the highest scored sentence from the second most important concept, and so forth in decreasing order of concept importance. Thus, each selected sentence describes a different concept and the most important concepts are captured in the summary. The other methods described by Ozoy et al. often deal with choosing multiple sentences from the same concept, which is not desirable in this context because our summaries are fairly short.

3.2 Abstractive Summarization

Abstractive summarization techniques are much harder to train because they involve modeling language well enough to produce both grammatically correct English sentences and sentences that summarize the inputted article text well. This requires a huge amount of training data and time. We therefore use a pre-trained model: Google's Text-to-Text Transfer Transformer Model (T5) introduced in a paper by Raffel et al.[4] T5 is an encoder-decoder model with bidirectional, multi-head self-attention layers. It is a relatively new model that is based on BERT with a few modifications. It was trained on the Colossal

[3] Ozsoy, Makbule & Alpaslan, Ferda & Cicekli, Ilyas. (2011). Text summarization using Latent Semantic Analysis. Journal of Information Science. 37. 405-417. https://doi.org/10.1177/0165551511408848.

[4] Raffel et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research. 21 (140). 1-67.

Clean Crawled Corpus (C4), which is 750 Gigabytes for scraped webpage data. It has also been fine-tuned to specific types of tasks such as translation and, relevant to us, summarization. For summarization, it encodes an article and uses it to produce a summary of that article.

The T5 implementation has two limitations. First, it is only able to encode a maximum of 1,017 tokens of an article because the self-attention layer considers the entire sequence and thus requires n^2 calculations where n is the number of tokens. If a topic appears for the first time late in an article, it may not be represented in the summary because it was never encoded. However, we do not believe this will be a significant issue for the CNN corpus. Only 15% of the articles have more than 1,017 tokens and the maximum number of tokens is 1,819. Additionally, as previously discussed, news articles usually present highlights in the first several sentences, followed by supporting details.

The second limitation is that the generated summary may end in a partial sentence. The model can generate an end of sequence character to end the summary, but we also specify a minimum and maximum length according to our previously discussed heuristics, which can result in a summary reaching its maximum length before generating an end of sequence token. However, because we expect content creators to edit the produced summaries, this is also not a significant issue.

We explore two ways to decode summaries from T5. The first decoding technique is Greedy search, which, for each word at position t in the summary, selects the word with the highest probability given the context of all previously seen words in positions 1 to t-1. An alternative is Beam search, which iteratively selects the best *num beams* sequences at each step. This allows the decoding to select a sequence containing a high probability word that is after a low probability word. Beam search requires hyperparameter tuning and higher computation time. Given that higher computation time, for this report we use *num beams* equal to 5 because a grid search through potential values was computationally prohibitive.

Other decoding methods exist that use sampling methods to introduce randomness to the generated text to make it sound more human-like.[5] However, we want the summaries to closely follow the content in the article and thus do not want any randomization. For example, using a sampling method on an article about a man attacked by a tiger generated a summary that claimed he was “conscious and talking with the animal” in the ambulance.

For both decoding techniques, we specify a parameter that disallows n-grams of length four to repeat in the summary. Otherwise, the model tends to generate repetitive sequences of words such as “the heat index will make it feel like 113. The heat index will make it feel like 113”, as observed for one article about a heat wave. Length four was chosen because bigrams and trigrams can be reasonably repeated in entities such as “New York City”.

3.3 Training & Configurations

We have identified three extractive algorithms and one abstractive algorithm to train with our data and compare the results. Each of these algorithms also has a variety of hyperparameters and other configurations that we need to select. The size of the grid of all possible combinations of configurations is in the hundreds for some algorithms. Thus, for computation purposes, we train each possible configuration for each possible algorithm on one tenth of the total data (1,000 articles). We then evaluate each of these trained configurations with the three metrics described in the next section and select the best configuration for each metric. Finally, we train the three selected configurations on the

[5] Some of these methods are described in this blog from Hugging Face: https://huggingface.co/blog/how-to-generate

full dataset and evaluate, resulting in one fully trained model per evaluation metric per algorithm for a total of 12 trained models. We can then compare the performance of models optimized for different evaluation metrics and trained with different algorithms.

Table A summarizes the configuration options for each algorithm. We take the cross product of all options to get the set of all possible configurations for each algorithm.

All of the extractive algorithms share the same initial text cleaning steps: down-case all words, remove all punctuation, and drop all sentences with only one word in them (usually due to poor sentence tokenization). Beyond this basic cleaning, TextRank and LSA perform other cleaning steps as part of their potential configurations including stop word removal, lemmatization, and stemming.[6] TF-IDF is treated as a baseline model and thus we do not use additional text cleaning for that model. No text cleaning is required for the abstractive T5 models because the encoder takes features like capitalization and punctuation into account.

TextRank and LSA both require vectorizing the article sentences. Vectorization options include a Bag of Words model that represents words with a count in the sentence, a binary indicator, or a normalized count or indicator using term frequency or TF-IDF. TextRank can also use GloVe or FastText embeddings to calculate the average embedded representation of all words in each sentence. The advantage of FastText is that it uses word parts to generate embeddings for Out-of-Vocabulary words, at the disadvantage of consuming a large amount of RAM.[7] We also consider using different combinations of unigrams, bigrams, and trigrams as tokens for both LSA and TextRank.

TextRank must also choose a similarity metric with which to compare sentence vectors. Options include cosine similarity, Jaccard similarity, and Hamming similarity. Jaccard calculates the percent of words in two sentences that are the same and Hamming calculates, out of all words in the vocabulary, what percent of words are both in or both not in two sentences. Both metrics only make sense on binary bag of words vectorizations.

As previously discussed, the primary configuration choice for the abstractive T5 algorithm is greedy versus beam decoding. Otherwise, it does not require many of the other steps such as additional text cleaning or sentence vector representation.

Finally, all models use a heuristic to control the length of the predicted summaries. These heuristics are based on the exploratory analysis in Section 2.1 that found that an average gold standard summary has 1 sentence for every 8 article sentences and 1 word for every 12 article words. For the extractive algorithms, we apply the sentences heuristic by selecting the best scored article sentences until we reach ceiling(number of article sentences / 8). We apply the words heuristic by counting words in the selected best scored article sentences until either the next sentence would exceed ceiling(number of words in article / 12) (“strict”) or this number was reached in the last sentence (“lenient”). Meanwhile, the extractive algorithm can only apply the words heuristic because the model takes parameters for the maximum and minimum length of the generated summary in terms of tokens. We set the minimum to ceiling(number of words in article / 12) – 12 so that the generated summary is not significantly shorter than the heuristic, but the decoder can generate an end of sequence token early. We set the maximum

[6] The models produce the original un-cleaned version of these sentences for the predicted summaries.

[7] In order for the FastText embeddings to fit into memory, we had to compress the embeddings by dropping less popular words. We used the compress-fastText package (https://github.com/avidale/compress-fasttext), inspired by a blog by Vasnestov Andrey: https://medium.com/@vasnetsov93/shrinking-fasttext-embeddings-so-that-it-fits-google-colab-cd59ab75959e.

to exactly the heuristic quantity in the strict version and to the heuristic quantity + 12 in the lenient version.

4. Evaluation

4.1 Evaluation Metrics

We use ROUGE (Recall-Oriented Understudy for Gisting Evaluations) metrics to evaluate the auto-summarization models. These metrics look at the overlap between tokens in the gold standard summary and the predicted summary. This is admittedly an imperfect metric because the predicted summary can have the same semantic meaning as the gold standard summary without using the exact same words. Using ROUGE metrics is industry standard, but developing metrics that consider semantic meaning is an area for further research.

We use three specific metrics: recall, precision, and F1 score. Recall is the number of overlapping tokens divided by the number of tokens in the gold standard summary. A low recall indicates that the predicted summary did not include many of the words in the gold standard summary. Precision is the number of overlapping tokens divided by the number of tokens in the predicted summary. A low precision indicates that the predicted summary includes a lot of extra information not in the gold standard. Precision and recall are in opposition because a very long predicted summary could theoretically have prefect recall at the cost of poor precision. F1 is the harmonic mean between precision and recall.

All these metrics are calculated where the token is a unigram. We also evaluated all the models using an average of each metric calculated with unigram tokens and bigram tokens. Bigrams can capture phrases in common between the predicted and gold standard summaries. However, the configurations selected as best for the unigram and blended unigram-bigram metrics are almost always the same or very similar. Thus, we only consider the unigram metrics for simplicity.

4.1 Best Configurations for Each Model, Evaluation Metric

Table B lists the best configurations for each model that maximizes each evaluation metric. TextRank always selects binary bag of words vector representations, indicating that for calculating sentence similarity, counts do not give any useful information and, in fact, may be misleading as a word appearing an additional time does not have the same importance as a word appearing versus not appearing. However, count bag of words is used for LSA precision and recall, indicating that the count of words is indicative when finding latent topics/concepts in a document. TF-IDF normalization is not used except for the TextRank recall-optimized configuration. Interestingly, TextRank never selects a model that uses an embedding for the vector representation. This is likely because it is difficult to represent sentences with word embeddings because taking an average across all words obscures the sentence's meaning.

All three abstractive configurations use Beam search, which makes sense as it is more likely to find a globally optimal solution than greedy at the cost of computation time. Finally, the length heuristic chosen is very consistent across algorithms for each evaluation metric: all precision-optimized models select the "strict" words heuristic, all recall-optimized select the sentences heuristic, and all F1-optimized select the "lenient" words heuristic. As discussed in Section 4.3, these heuristics correspond with relative summary length such that precision-optimized summaries are shorter and recall-optimized summaries are longer.

4.2 Quantitative Comparison: Evaluation Metrics

After training the best model for each algorithm and each evaluation metric, we have 9 extractive models and 3 abstractive models.[8] Figure 5 compares each model for its targeted maximized metric averaged across all articles. LSA consistently performs the worst for all metrics and TextRank performs the best out of the extractive models. TextRank's recall-optimized model especially performs significantly better than the other recall models, including the abstractive model; on average almost half of the words in a gold standard summary appear in its predicted summary.

Given our three goals in the Introduction, recall is a priori assumed to be the most important metric so that the content managers have enough information about the article to work with. For example, TextRank optimized for precision generates the following short, vague summary of an article about the carbon footprint of football teams: "The cistern stores rainwater that can be used to irrigate the pitch, but compared to most clubs' water consumption, it is just a drop in the ocean". The recall model gives more details about the current carbon footprint and how the club's efforts are paying off.

The abstractive T5 model optimized for precision performs significantly better than the other precision models, but the T5 model optimized for recall performs significantly worse than the other recall models. This is likely because the T5 model produces shorter summaries than the recall-optimized extractive models, both because it can only use the words heuristic to control summary length, which generally produces shorter summaries than the sentences heuristic, and because it can generate an end-of-sequence token before reaching the heuristic length. However, abstractive models are able to synthesize multiple article sentences into one summary sentence while extractive models must use the entire article sentence, even if it contains unnecessary information. Thus, a shorter summary may be acceptable or even desired. Further investigation is needed.

It is important to note that all of the differences in metrics between extractive models in Figure 5 are significant.[9] We tested significance using a paired bootstrap test as follows: calculate the difference in metric performance between a pair of models, generate 50 bootstrapped samples,[10] train both models on each sample and calculate the difference in metric performance, and finally calculate the p-value as the percent of replicated differences that are at least two times larger than the original difference. We compare to two times the original difference because the Null hypothesis is under the assumption of our biased data that produced the original difference.

Figure 6 expands the view of Figure 5 by plotting all three metrics for each of the 12 models. Even if a model performs very well for its targeted optimized metric, it may perform very poorly in another (in fact, models that perform the best in recall will, by definition, perform the worst in precision). The TextRank recall-optimized model does perform poorly in precision compared to the other models, but the degradation is reasonable. On average 20% of the words in the predicted summary appear in the gold standard summary, so the model is not producing a large amount of extra information in order to increase recall, thus following our third goal: "do not provide so much information that the task of editing down the summary is equivalent to producing a summary from scratch." If we wanted to achieve a different balance between precision and recall, we could optimize for an F-measure weighted more

[8] More accurately, two abstractive models because the selected configuration for F1 and recall are the same.
[9] We did not test significance between the extractive and abstractive models because the differences are larger than those between extractive models, which proved to be significant, and because the decoding time for the abstractive models was so large that decoding many bootstrap samples was unrealistic.
[10] Ideally, we would have constructed more bootstrap samples, but given model training times, this was not feasible.

unevenly towards recall. However, the precision of the recall-optimized model is high enough that this is not necessary.

Finally, we implemented a very simple baseline extractive model to compare our results against. The baseline uses the first several article sentences, up to the heuristic lengths, as the predicted summary. This simple approach performs comparably to recall-optimized TextRank with an average recall of 0.470 versus TextRank's 0.458. The success of this baseline is due to the news article's Pyramid structure where the main concepts are typically summarized in the first few sentences. Meanwhile, TextRank is more broadly applicable to a wider range of documents with a variety of structures. As previously stated, consulting materials are often written using the Pyramid Principle, but that corpus is also more diverse than the CNN corpus and often deals with more complex topics. We would ideally like to use this summary tool to generate summaries for a wide variety of materials such as client proposals, case work, trainings, internal documents etcetera. Thus, we do not expect this baseline model to be sufficient and instead focus on more robust models that consider the entire article text when generating a summary.

4.3 Quantitative Comparison: Summary Length

Beyond the evaluation metrics, it is also informative to examine the relative lengths of the summaries produced by each model. Figures 7 and 8 show that recall-optimized models on average produce longer summaries and precision-optimized models produce shorter summaries, as is expected by the definition of precision and recall. This is largely controlled by the choice of length heuristic. Again, longer summaries are desirable in this context if precision remains reasonable.

Summaries produced by the abstractive model are shorter, which, as previously discussed, could be acceptable if the summaries are more too-the-point than extractive summaries. This is supported by Figure 9, which shows that abstractive summary sentences are shorter on average than extractive summary sentences that are sourced from the article and thus often contain extra information. However, Figure 10 reveals that even recall-optimized abstractive summaries are even shorter than the article's gold standard summary. These findings taken together suggest that abstractive summaries are too short and may miss some key information in the article.

Based on these quantitative comparisons in the management consulting context, TextRank recall-optimized is the best extractive model. There is some evidence that the T5 abstractive model produces insufficient summaries, but we need to further examine them in comparison to the TextRank recall model. Qualitative examination is needed.

4.4 Qualitative Comparison

The ultimate test of model quality is to evaluate the outputted suggestions our innate human understanding of language. Given the quantitative evidence, we will focus on the TextRank recall-optimized and abstractive recall-optimized models.[11]

Section 3 discussed several theoretical limitations of abstractive models. One of the biggest concerns is that abstractive summaries could generate information not originally in the article. This is rare, but there are some instances of misinformation in the T5 predicted summaries. For example, the predicted summary for an article about George Zimmerman and fears of parents of Black children contains the sentence, "if Zimmerman was black, he wouldn't have been the parent of a black man". This obviously makes no sense and could be misleading. Another example is a summary about a kidnapping that contains the sentence, "he says the family is receiving 'exceptional cooperation' from the united states

[11] The recall-optimized abstractive model best mitigates the concern of abstractive summaries being too short.

and the united states". In reality, it was the Cuban government being cooperative. This sentence also contains a repetitive phrase ("the united states and the united states").

Additionally, when we directly compare extractive and abstractive summaries for the same article, there are many instances in which the abstractive summary misses key details about the article. For example, TextRank summarizes an article about Dick Cheney being released from the hospital after a heart transplant as, "Former Vice President Dick Cheney has been released from a Virginia hospital 10 days after undergoing a heart transplant, his office said Tuesday." The T5 summary is "former vice president was on the cardiac transplant list for more than 20 months. he had a left ventricular assis". This summary does not discuss the success of the heart transplant, which is the primary point of the article. Similarly, the TextRank summary of the previously mentioned George Zimmerman article discusses how parents of young, Black men feel afraid for their children in the wake of the Zimmerman trial. The T5 summary gives a hint of the article's theme but doesn't discuss its main points: "john avlon: if Zimmerman was black, he wouldn't have been the parent of a young black man. he says he's a parent of young black males in the middle-class, predominantly white neighborhood."

There are also several less concerning issues observed with the abstractive summaries that could be easily corrected by a human editor. For example, capitalization is often not recovered correctly, as seen in the sentence "jeb Bush is the clear Republican presidential frontrunner". Also, sentences are often cut-off at the end of a summary due to the maximum words parameter.

There are also examples that show the advantages of abstractive summaries over extractive. They are often more too-the-point than extractive summaries and can convey the same information in fewer words. The extractive summary of the article about football clubs' carbon footprint includes minutia about different opinions on how many liters of water it takes to irrigate a pitch and how a rainwater cistern is used. The T5 summary quickly describes the main points: "a 2008 survey by "Ethical Consumer" looked at the eco-credentials of clubs in the english premier league. the club says it has reduced landfill by 85 percent, moved to electric vehicles at the ground, and used eco-friendly paper for match-day programs. it's possible for small clubs to do their bit, but their use of solar". Another advantage of abstractive is that it takes the order of sentences into account. The extractive summary about a storm in Sardinia starts with the sentence "But Vargiu said a lot of ministry staff and emergency workers had come to aid the town, and there were many volunteers helping its inhabitants." The abstractive summary starts with "a storm has killed at least 16 people on the Italian island of Sardinia."

The most concerning features of abstractive T5 summarization highlighted by qualitative examination are that they can contain false information and do not always contain the main points of the articles. Extractive TextRank summarization often produces overly long, complicated summaries, but the summaries are guaranteed to produce correct summaries that can be edited down.

5. Recommendation

In the context of creating an auto-summarization tool for content creators at a management consulting firm to generate draft summaries, TextRank optimized for recall is the best model. It produces summaries that consistently capture the main points of the article, even if they are too long, contain extra detail, and have sometimes illogically ordered sentences.

The three stated goals for this report were: (1) capture the main themes of the content, (2) preferentially provide too much information over too little information, (3) but do not provide so much information that the task of editing down the summary is equivalent to producing a summary from scratch. The weakest point for recall-optimized TextRank is the third goal, but the length heuristics place

a limit on the summary length and thus the precision scores are reasonable. Meanwhile, abstractive T5 summarization does less well on the first two points.

The major advantages of abstractive summarization, creating succinct summaries without extra details and considering sentence order, would be important if we wanted to generate summaries with no or very little human review. We could also try to force the T5 model to produce longer summaries by further tuning parameters and developing new length heuristics, but even so, given the other concerns about generating incorrect information and model explainability, extractive summaries would still be preferable in this context given that we have human editors.

Model explainability is very important. The users of this tool will on average have no experience in machine learning and will likely have low trust of a black box abstractive model that produces summaries seemingly by magic. Many content creators would spend a lot of time reviewing the original document to ensure that the abstractive summary is accurate. Meanwhile, extractive summarization will return sentences that they themselves wrote. Thus, driving adoption of an extractive tool will be easier.

6. Further Research

There are many opportunities for continued work and research. Evaluation for auto-summarization is usually based on token overlap, but evaluation using semantic similarities would provide a more robust view of the accuracy of generated summaries. For extractive summarization, other methods exist that we did not explore involving weighted scores or classification on features such as if a word in a sentence appears in the title, the position of a word in a sentence etcetera.

There are many possible extensions to the abstractive model. Parallelizing the decoding process would allow us to perform grid search through hyperparameters like number of beams and to generate summaries for a larger set of articles. An alternative model to T5 is the Longformer encoder-decoder proposed by Beltagy et al.,[12] which can encode up to 16,000 inputted tokens, as opposed to T5's 1,017 tokens. However, Longformer's decoding time is longer than T5's and, as previously discussed, the 1,017 tokens were not a significant limitation in our case. Finally, we could fine-tune a pre-trained encoder-decoder model to our specific corpus. This is likely not necessary for the CNN articles because they use typical English concepts and sentence construction, but for a more specific corpus like that of a management consulting firm that deals mostly in PowerPoint presentations, fine-tuning may be necessary for good performance.

Finally, a possible way to combine the benefits of both extractive and abstractive methods is to extract key phrases rather than sentences from the article text and combine those phrases into sentences. This would ensure that all phrases are directly from the article, but would enable us to combine ideas from multiple article sentences more succinctly. Some language modeling would likely be required to combine phrases in a logical manner. These key phrases could alternatively be used for other meta-data tagging beyond summaries.

In sum, auto-summarization is a powerful tool that can revolutionize a manual content management system by creating significant time-savings for content creators and thereby increasing the quality and quantity of created meta-data. When paired with human editing, recall-optimized extractive summarization produces summaries with sufficient information for the editor to quickly create a well-formed summary.

---

[12] Beltagy, Iz & Peters, Matthew & Cohan, Arman. (2020). Longformer: The Long-Document Transformer. https://arxiv.org/abs/2004.05150.

Appendix

Figure 1: Histograms of length of articles and gold standard summaries

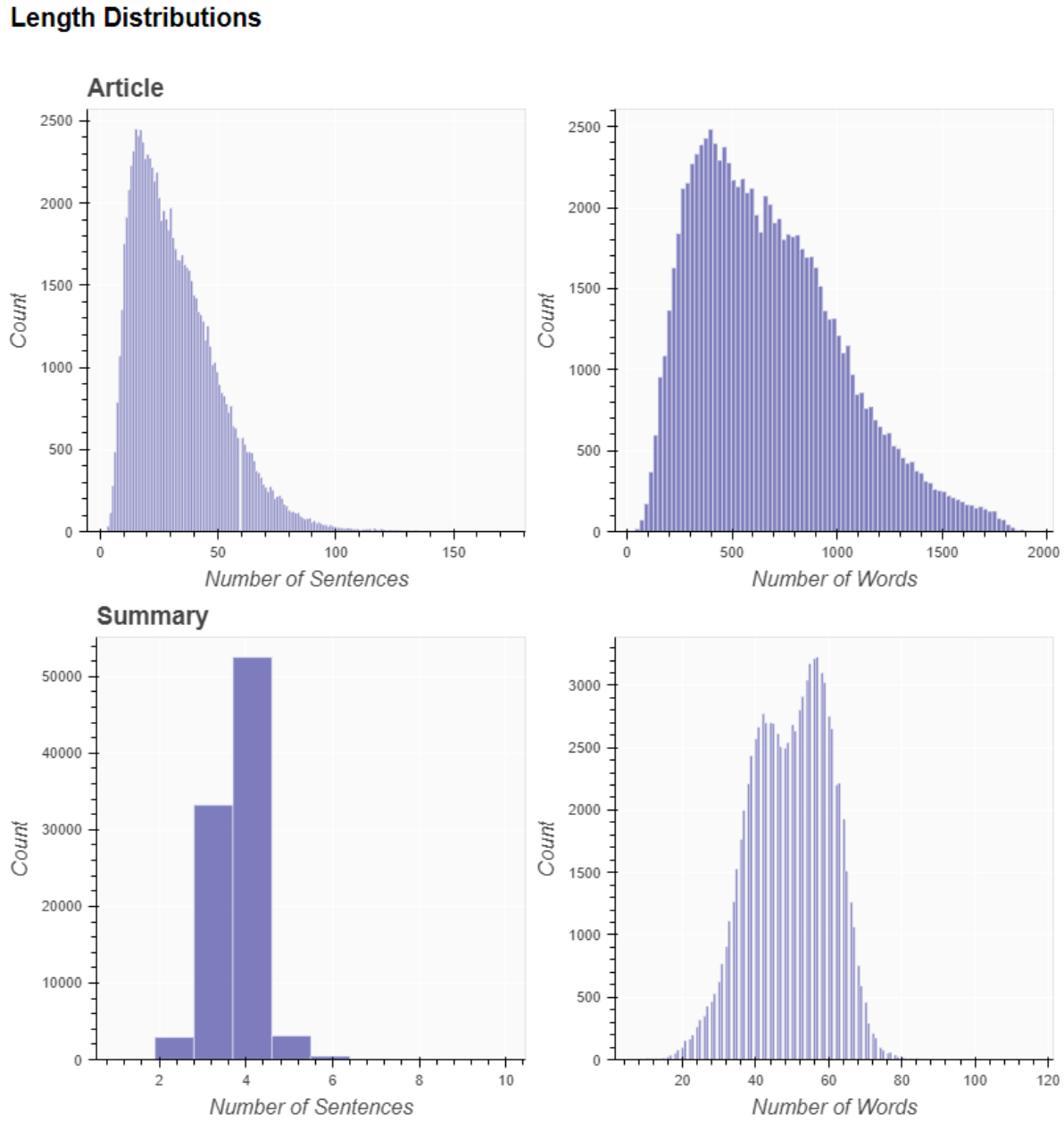

Figure 2: Histograms of number of article words/sentences per gold standard summary words/sentences

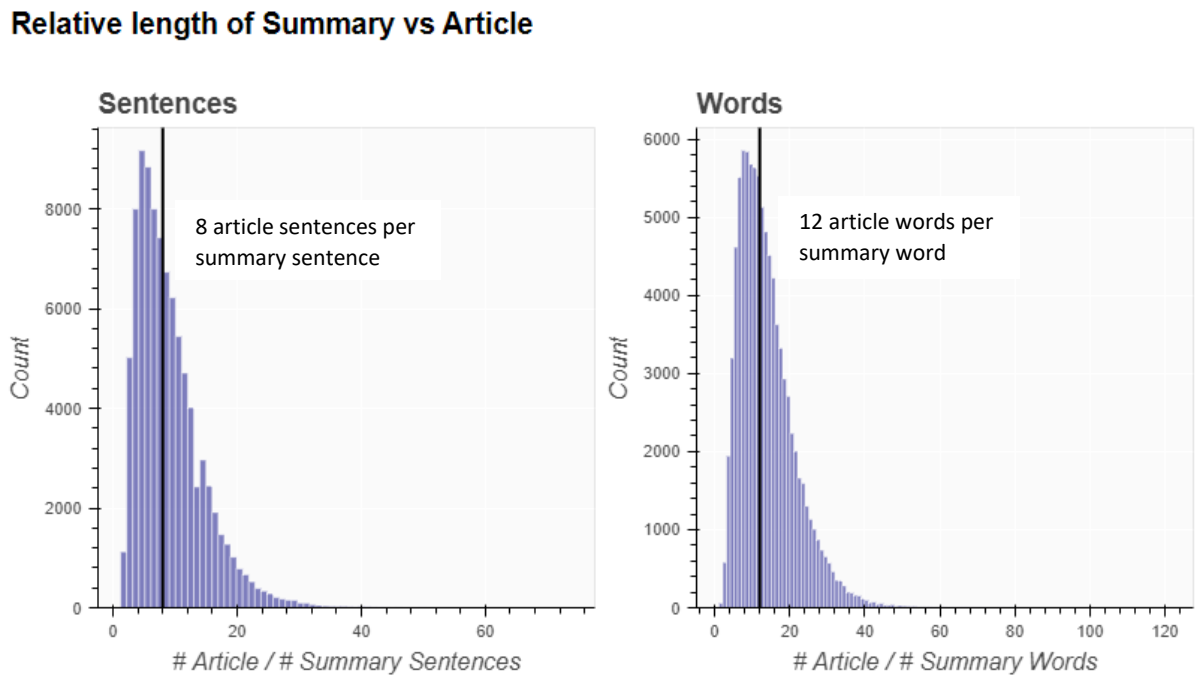

Figure 3: Histogram of relative length in words of article sentences versus gold standard summary sentences

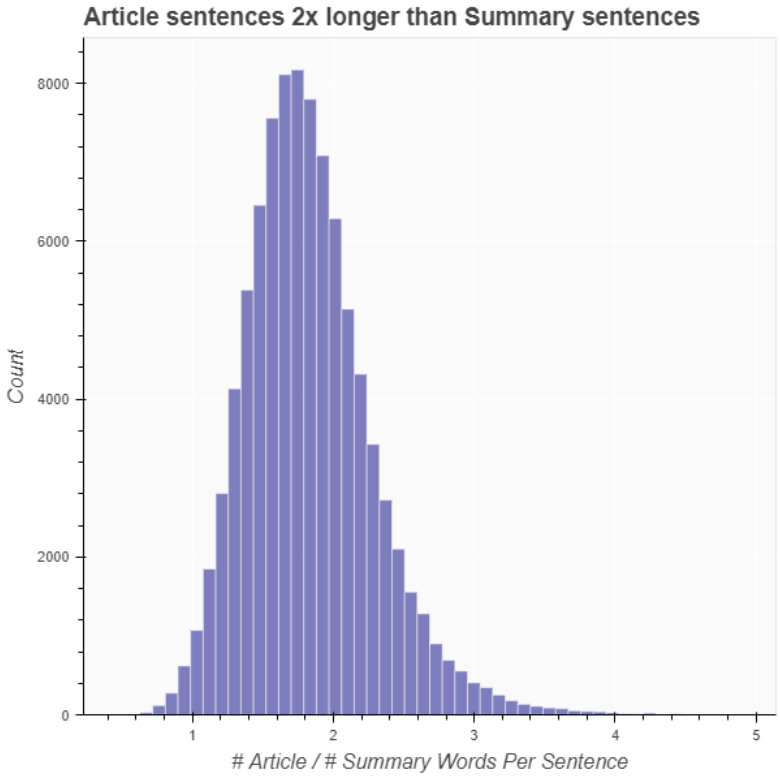

Figure 4: Histograms of words in gold standard summaries that are not in the article

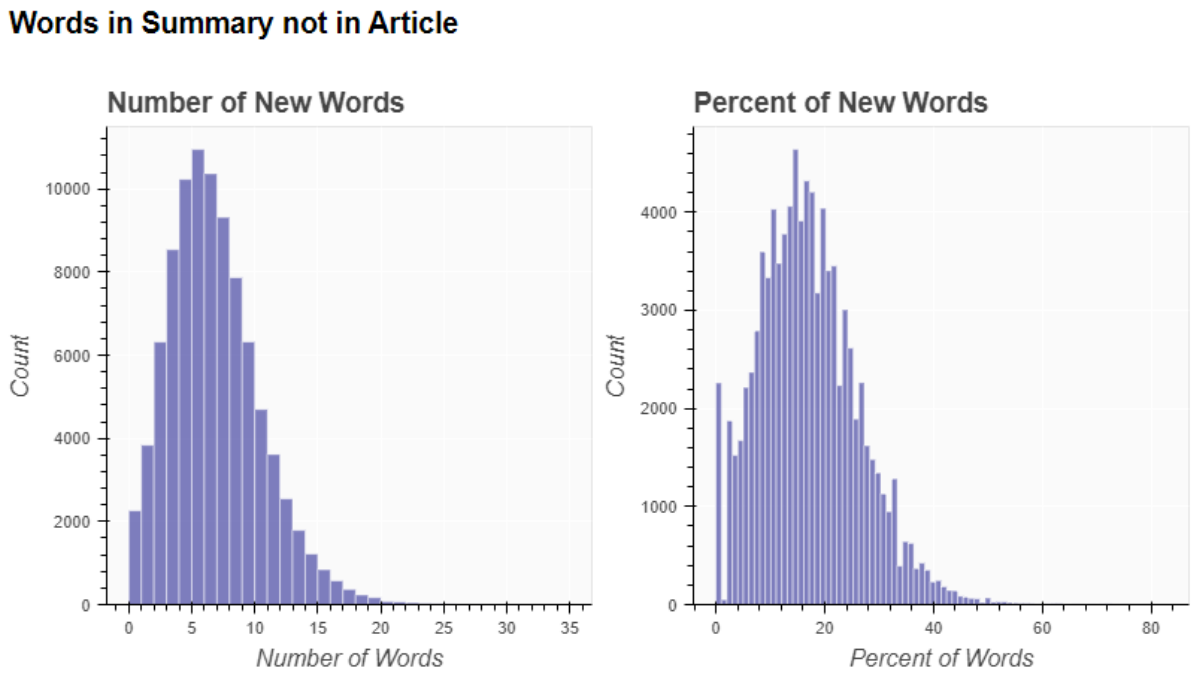

Figure 5: Performance of 12 trained models by their optimized target metric, averaged across all articles

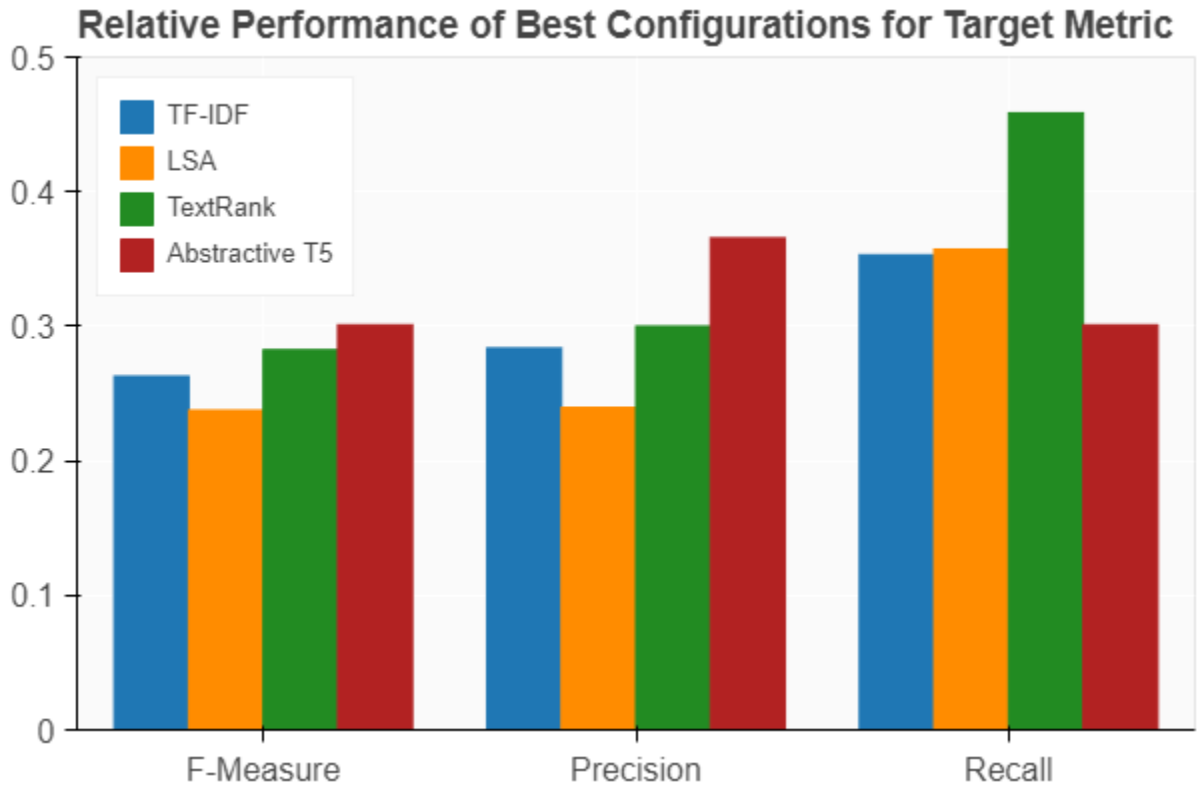

Figure 6: Performance of 12 trained models by all 3 evaluation metrics, averaged across all articles

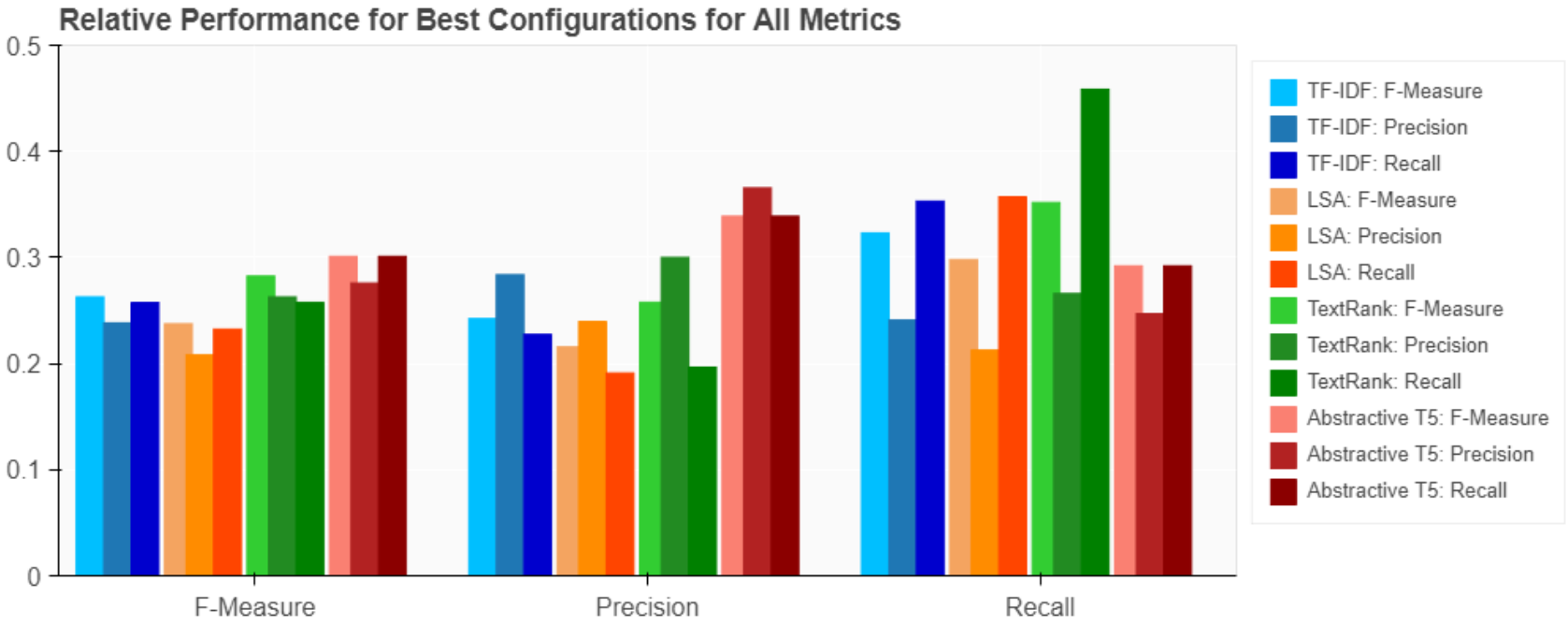

Figure 7: Predicted summary length by average number of sentences

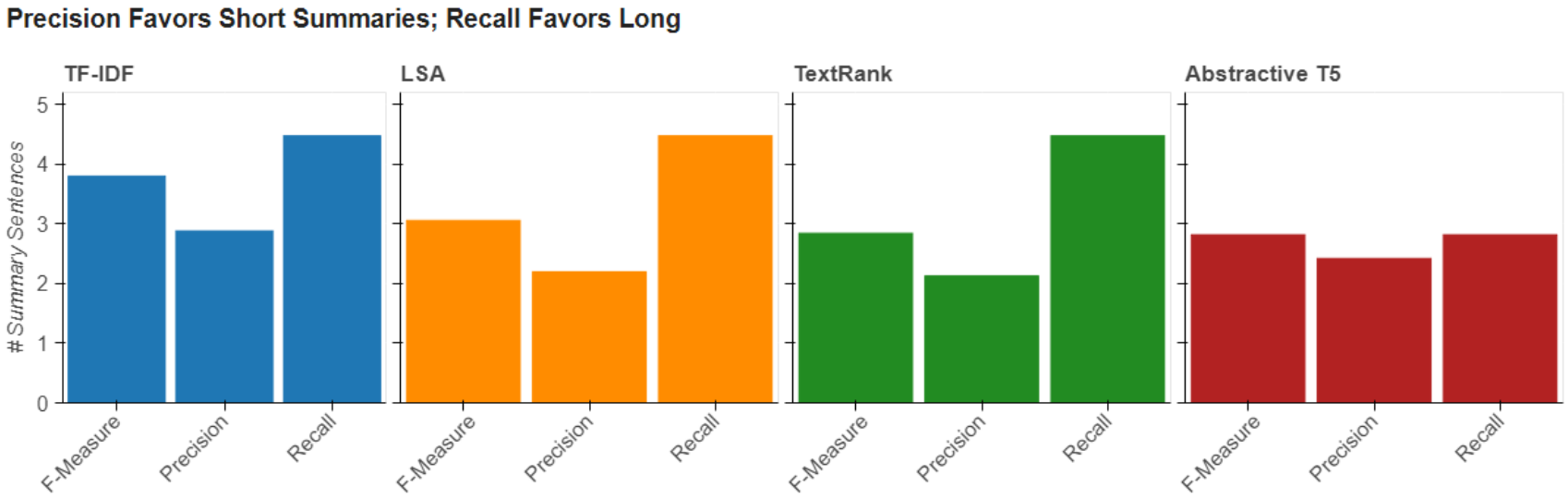

Figure 8: Predicted summary length by average number of words

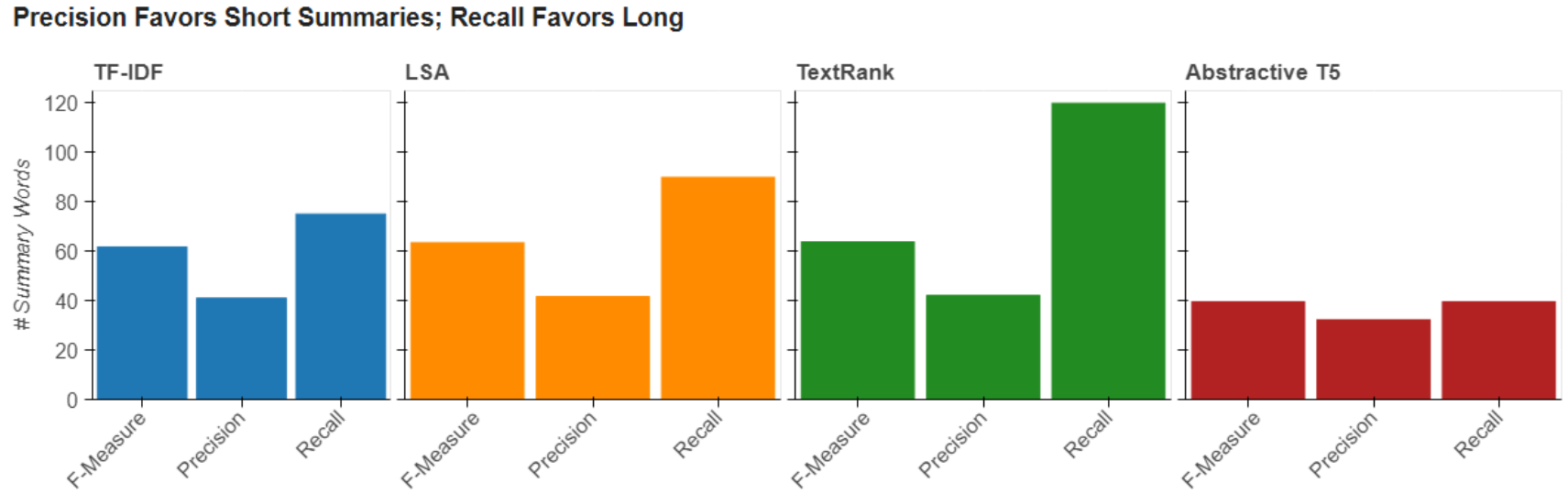

Figure 9: Average number of words per predicted summary sentence

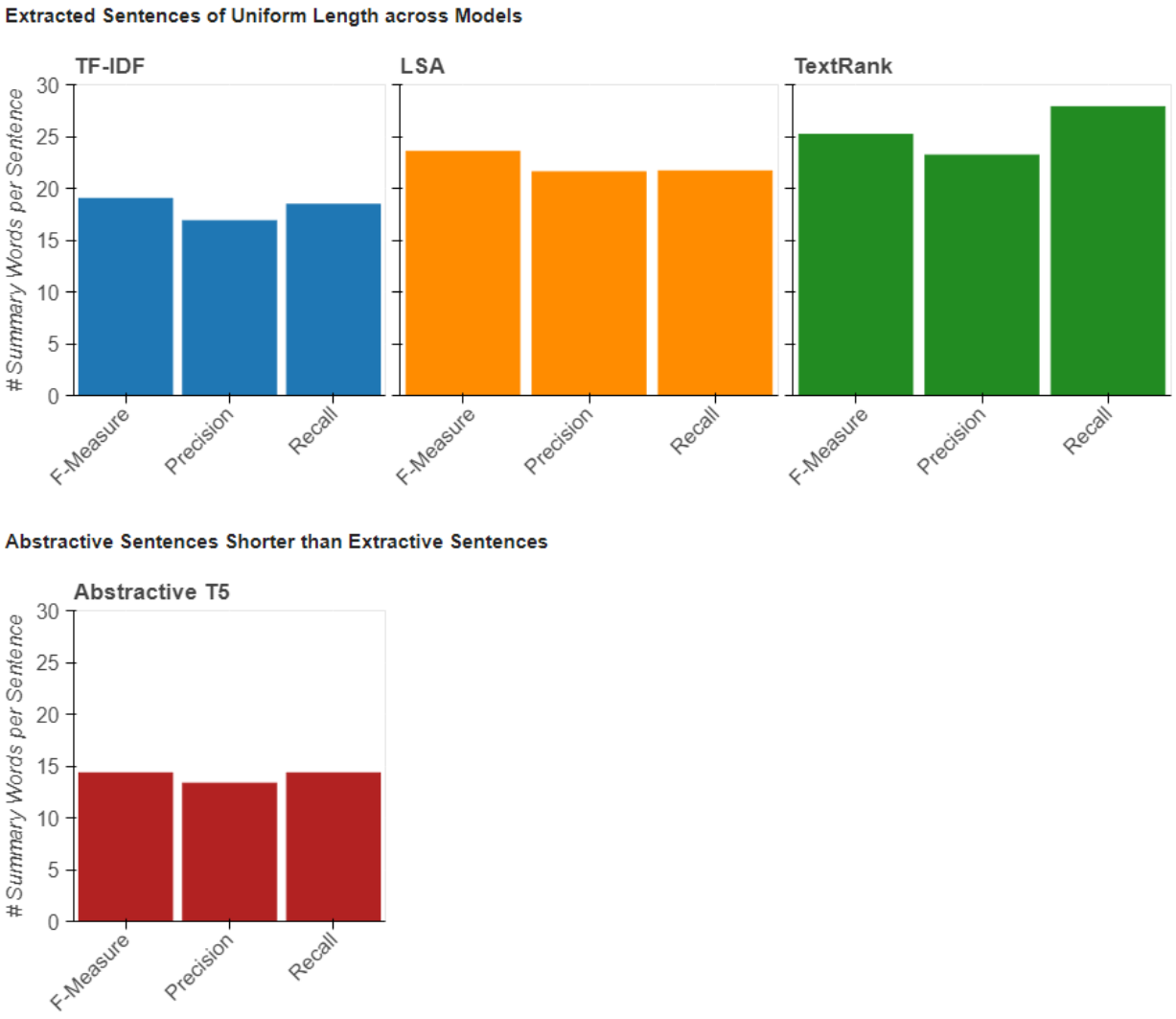

Figure 10: Average difference in number of words between predicted and gold standard summary

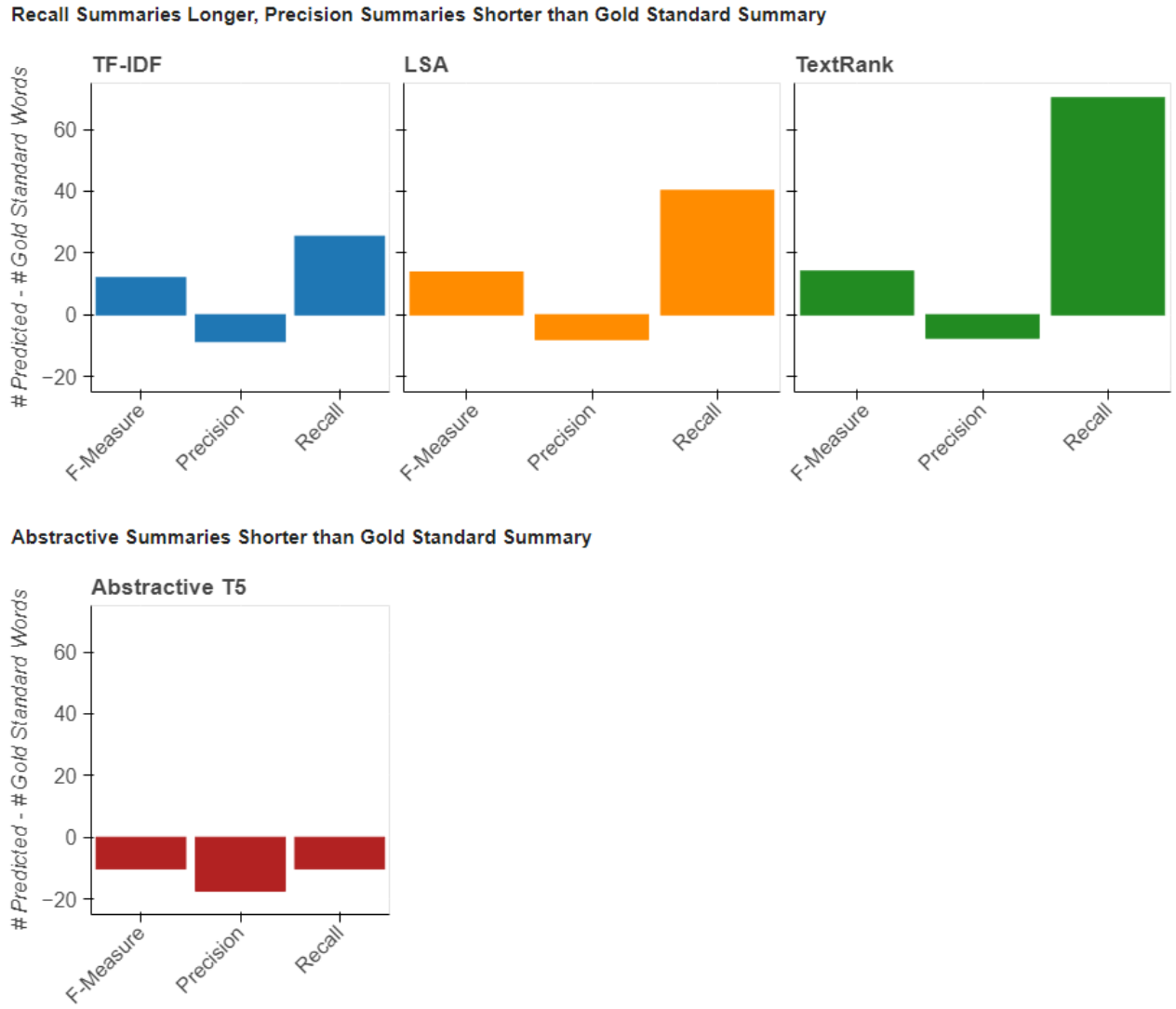

Table A: Configuration Options

	Text Cleaning	**Vectorization**	**Included N-Grams**	**Similarity Metric**	**Summary Length Heuristic**
TextRank	• Remove vs. keep stop words • Original words vs. lemmatize vs. stem words	Bag of Words: a. counts vs. binary b. no normalization vs term frequency vs TF-IDF OR Embeddings: a. GloVE b. FastText	• Unigrams • Unigrams + Bigrams • Unigrams + Trigrams • Unigrams + Bigrams + Trigrams	• Cosine similarity • Jaccard similarity • Hamming similarity Jaccard and Hamming only used for binary bag of words vector representation	• 1 summary sentence per 8 article sentences • 1 summary word per 12 article words: strict • 1 summary word per 12 article words: lenient
LSA	Same as TextRank	Bag of Words: a. counts vs. binary b. no normalization vs term frequency vs TF-IDF	Same as TextRank	N/A	Same as TextRank
TF-IDF	• Keep stop words and original words	N/A	N/A	N/A	Same as TextRank
T5 (Greedy/ Beam)	N/A	N/A	N/A	N/A	1 summary word per 12 text words • Strict • Lenient

Table B: Best Configurations

	Precision	Recall	F1
TextRank	• Remove stop words • Lemmatize • Binary unnormalized Bag of Words • Unigram + bigram + trigram tokens • Cosine similarity • Strict words heuristic	• Keep stop words • Stem • Binary TF-IDF Bag of Words • Unigram tokens • Cosine similarity • Sentences heuristic	• Remove stop words • Stem • Binary unnormalized Bag of Words • Unigram tokens • Jaccard similarity • Lenient words heuristic
LSA	• Remove stop words • Stem • Counts unnormalized Bag of Words • Unigram + trigram tokens • Strict words heuristic	• Keep stop words • Stem • Binary unnormalized Bag of Words • Unigram + bigram + trigram tokens • Sentences heuristic	• Keep stop words • Lemmatize • Counts unnormalized Bag of Words • Unigram + trigram tokens • Lenient words heuristic
TF-IDF	• Strict words heuristic	• Sentences heuristic	• Lenient words heuristic
T5	• Beam search • Strict words heuristic	• Beam search • Lenient words heuristic	• Beam search • Lenient words heuristic