Joshua Norris

Dr. Ferrer

Artificial Intelligence

7 December 2017

Text Summarization

For humans, time is the only thing that is truly priceless. Almost all areas of advancement are motivated by giving us more time. We made cars so that we could spend less time traveling. We made phones so we could spend less time going to see a person face to face. We made calculators so that we no longer have to calculate long division and complicated multiplication. There is not a field that is quite as motivated by the idea of giving the user more time than the field of computing. Text summarization is one of the best ways that the field of computing does this for the user. Multi-document text summarization is defined as, “an automatic process where the essential information is extracted from the multiple input documents” (source 1). Summarization is done in two ways: extraction-based summarization, and abstraction-based summarization. The role of extraction-based summarization is that it “selects some original sentences from the source documents to create a short summary [of the document.]” The role of abstraction-based summarization is that it “can construct a sentence whose fragments come from different source sentences.” In Layman’s terms, text summarization takes a document or a set of documents and then creates a summary of the important information. In the following paper I will talk about a supervised extraction-based summarizer, an unsupervised extraction-based summarizer, a classifier used for supervised learning in both extraction and abstraction based summarizers, and an unsupervised abstraction-based summarizer. I will conclude talking about the applications of text summarization and the differences between extraction and abstraction, and a supervised and unsupervised learner.

The first algorithm I will discuss is the supervised approach to extractive text summarization on the Naive Bayesian Classifier using Timestep Strategy. This algorithm has a seven step process:

1. Set of documents is provided for processing.
2. From the set of documents, frequently used related documents are selected by the system for processing.
3. Preprocessing work is done and the sentences are broken into words.
4. Score is calculated for each word using Bayesian Classifier.
5. Score is calculated for each sentence.
6. For each sentence group, one sentence level illustration is selected.
7. The sentence level illustration is either generated as a linear sentence of further compressed if necessary.

The first step is semi-obvious, the set of documents is provided and with this set of documents we are also given how many times each document has been visited by the user. Also at this point in the algorithm it is implied that the Naive Bayes has already been trained on a set of similar documents. For the second step what happens is the algorithm only selects a portion of the input documents to be processed, for instance if 10 documents are inputted, then the 2 (this number is arbitrary the reference does not imply one percentage or number that is better than the others) related documents that have been visited most by the user is then processed. The third step is again self explanatory, it is fed into a Natural Language Processing software that will split the document into sentences and words using things like spaces, tabs, and periods. The Score value is where this gets interesting. First the Term frequency value (the number of times the word appears / the number of words in the document) and the Inverse Document frequency is calculated (Log(total number of documents / total number of documents with the word in it) and then they are multiplied together in order to get the value T. This value is then used in the Naive Bayesian classifier in order to identify whether or not the given word is a keyword or not. The equation is below:

In this formula simply means given the Term value, the Document, the position value of the term and the position value of the sentence what is the probability that key is true. Then the next term, , simply means that given that it is a key, what is the probability that it has that given value. The T value is simply the TF times the IDF as specified above, the R simply means that document, the PT means the position of the term in the sentence, and the PS means the position of the sentence in the document. The position of the term in the sentence is simply in order of the occurrence in the sentence, but the PS is calculated using the formula listed below:

When the n is the number of sentences in the document, and the i is the order in which it occurred (the first sentence would have an i value of 1 and the second sentence, 2, etc.) and the C max value is the best centroid value out of all of the sentences in this document (further explanation of this will come below). This value is made to value the sentence that is closest to the start of the document. This is under the premise that the topic sentences which are typically one of the first sentences is what we want to choose. There is another approach to this where we do the same thing, but we also value the closest the sentence is to the end as well, so the lowest value will be the sentence in the middle with the highest being the last (which targets the conclusion) and the first (which targets the introduction). For the following step in the algorithm it scores the sentences individually. This is done by using the formula displayed below:

The wc, wp, and wf variables are simply weights given which are set to a constant to weigh each factor differently. The C value is the centroid of the sentence, this formula it listed below.

The TF is the term frequency of the specified word in the sentence as defined above. The IDF is the inverse document frequency as defined by above. And the R value is the number of documents that contain this word other that this specific document. This value is then added up for all of the words in the sentence. This variable is purposed for taking into account the importance of each word used in the sentence. The P value is the same value as the PS value used in the Naive Bayesian classifier above. The F value is found using the formula listed below.

This formula is used to find the similarity between this sentence and the first sentence in the document. Again this is focused on the premise that the most important sentence is typically the first sentence in the document. This formula can be adjusted to account for more than one sentence at the beginning of the document, or to include the end of the document to ge the conclusion portions. For step six, the algorithm selects the sentences for the summary using the algorithm below.

|  |  |
| --- | --- |
| (1) For all the sentences in the cluster | |
| (2) Begin | |
| (a) Sort the sentence in the descending order depending on the obtained score values after | |
| the reduction of the redundancy penalty. | |
| (3) End | |
| (4) Begin | |
| (a) Get the compression rate from the user | |
| (b) Select the required number of sentences based on the compression rate. | |
| (c) Sort the sentences in the ascending order depending on the timestamps | |
| (d) If the Timestamps are same | |
| (e) Begin | |
| (i) Compare the score values | |
| (ii) Sentence with the higher score value will appear first | |
| (f) End | |
| (g) End If | |
| (5) End | |

Using this algorithm above there is a set number of sentences the user desires for the summary and it simply takes that many sentences off of the cluster.

The second algorithm I will study is the unsupervised extraction algorithm called, TextRank. Text rank is a graph based algorithm that has a plot of vertices and uses each edge as a “vote” to decide importance. In other words, “the higher the votes the higher the importance”. More specifically we have a graph G = (V, E) which is a graph with a set of vertices and a set of edges which is a subset of V x V. These graphs can be either directed or undirected in practice. If it is directed then you will have a subset of vertices that are the current vertex’s parent nodes, and a set of vertices that this node points to. If it is an undirected graph, any node that points to it, the vertex points back. The score of a vertex is below.

Where In(Vi) is the parent nodes and Out(Vj) is the nodes that the sentence points to, and d is a damping factor between 0 and 1. The first step in creating the graph is to make a graph with the desired number of nodes all set to arbitrary values. I am going to look at TextRank used for sentence extraction, but using this algorithm it can be used to find keyword extraction, which usually consists of a supervised learner very similar to the algorithm I talked about in the above paragraph. For this implementation there will be one node for each sentence. The edges are then set for each of the nodes that need edges, and these edges are placed due to a “similarity” between the sentences. There are many ways to determine this, one of which is simply seeing which sentences have the most similar content. This can be calculated in multiple ways, one can use the number of common tokens , or count words of a certain syntactic category after going through syntactic filters. Whichever way is chosen, then there is a normalization factor that needs to be implemented. This is simply to account for the differences from sentence to sentence in length. This is done by dividing by the lengths of the two sentences. If two sentences are similar enough to pass a threshold that is determined by the user, the edges are places, and they are weighted based on the level of similarity between the two sentences. After this a simple ranking algorithm is run in order to rank all of the points depending on their score, and then the top ranked sentences are put together to create the summary.

The next algorithm I will look at is the Maximum Entropy Classifier used to perform supervised learning. As I studied text summarization I learned that the difference in the implementation of a supervised and unsupervised learner is very similar (see the two algorithms listed above). The difference between the two for this application is simply whether or not the creator of the algorithm wants to put a classifier in the program to help it gain better results. That being said, the Naive Bayesian classifier could be applied to both of the unsupervised learners I have mentioned. The same goes for the Maximum Entropy classifier. So because of this there is no specific algorithm designed only for the Maximum Entropy classifier, so I thought it would be more beneficial to explain what it is and talk about the several different ways it has been implemented. This algorithm is similar to Naive Bayes by the fact that is has an element and a label that goes with that element. The first thing that it does is calculate the probability distribution using the formula below.

[1]

The variable N is the size of the dataset so this is the same as the term frequency formula we looked at in the first algorithm. The next thing we need to do is have a formula for an indicator function as seen below.



This formula is simply a boolean that starts whether or not the given word exists in the document and if the document is labeled by the same label. We then take the element of a feature and find it with respect to the empirical distribution of P(n) as seen below.



What this formula does is simply find a value that takes into account the term frequency while ensuring that it exists in the class it is being compared to and adding these together for both x and y in order to get a value. We next want to add in the ability to use an additional statistic by using the formula stated below.

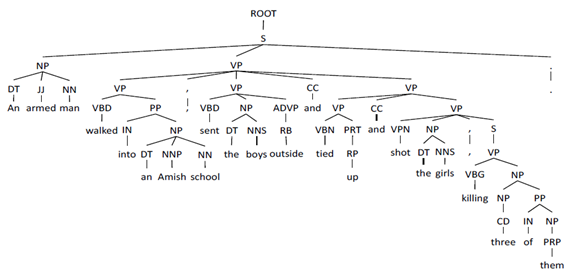


This formula is simply the term frequency of x in all the documents despite the label, times the probability of the label y given that x is true. Finally we want to focus the idea to fit the idea of Maximum Entropy by finding the value with the best entropy for the given task using the formula listed below.



This change in the formula is to focus our selection to pick the value that is closest to uniform in order to select the value with the best entropy. The value of log( p (y | x)) is added in in order to reward the value for having a high probability of selecting the label y for the word x. And because of this negative sign it means that the algorithm will select the value that scores the lowest on being uniform. What this allows us to do is search for terms that are important, but are not the most frequent term. This is the biggest difference between this classifier and Naive Bayes, the Naive Bayes expects the term to be most like the rest of the terms when the maximum entropy is looking for a term that is most unlike the others, but is still an important part of the document. It is very understandable why this algorithm is used so much in Text Summarization because it is very good at selecting phrases or sentences that are important to the semantics of the document, yet are different than the ones already accepted. This kind of classifier keeps us from being in a situation where we have a 10 sentence summary of a document and all 10 of the sentences are saying the same thing.

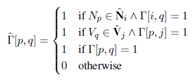
The final algorithm is an Abstractive Multi-document summarization via Phrase selection and merging. First the algorithm takes all of the sentences from each document and then creates constituency trees. Example Below.



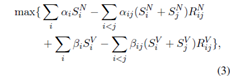
Next the algorithm extracts the NP (noun phrases) and VP (verb object phrases) phrases that are directly related to the sentence node. Along with these phrases, the algorithm will extract any VPs that are not a direct child of the sentence node, but is on a path of only VPs from the sentence node to the VP. After this the algorithm calculates the Salience Score. This salience score can be calculated using many different metrics, but this specific implementation uses a concept-based weight that incorporates position information. This is done by making a set of all of the unigrams, bigrams, and named entities. Next it uses the formula below to calculate a position based term frequency.



In this formula, the B value is the weight of the first paragraph and this value decreases as the paragraph gets further and further into the document. P is the position of the paragraph starting from 0. *P* is a constant that is positive, but less than 1. This formula is used to find the salience of a phrase as the summed weights of its concepts. Next the algorithm moves into the new sentence formulation part of the program. It makes a sentence by using one NP and at least one VP. To do this the algorithm must find compatibility between a NP and VP to ensure they are talking about similar things, and they make sense syntactically in a sentence. This is done first by using heuristics, and then it expands each cluster it makes by using coreference for NPs. this is just a coreference resolution package that clusters NPs from different places with the same or similar topics. Once we have formed these clusters, the algorithm chooses two, and these two serves as the other’s alternate. This then goes through a very similar process to cluster the VPs. In order to find out whether a NP and VP are compatible, it uses the formula below.



is only true when the noun (p) and the verb phrase (q) are in the same sentence. This is the third case for the above formula. The first and second case are if the NP or VP are in a cluster with a NP or VP that was in the same sentence as the NP or VP it is being compared to. Otherwise there is no compatibility. Next the algorithm uses Phrase-Based Content Optimization to optimize which phrases should be choses, the formula is below.



The alpha and beta values are selection indicators for the NP and VP. The S values are salience scores, and the R values are the similarity of pairs. This formula is made so that it penalizes similarity among the pairs to prevent redundancy. And then it selects the pairs for however many sentences it needs, combines them into a sentence and then outputs the summary. One of the important things to note about this algorithm is that it wants to make is so that NPs and VPs are treated differently to avoid the answer that we get in extraction summarization, which is just the sentences that were already in the document.

To compare the supervised and unsupervised learning in text summarization, it is very similar to what it is like comparing these two ways of learning in general. Supervised learning will learn faster, and assuming the dataset is good then it will most likely be more accurate as well, but there are obvious drawbacks, the biggest of which being that this needs a dataset to train on. And so while this works really well in practice for most real world applications that have to handle a wide range of topics and situations, constructing a reliable dataset becomes a very large task. For unsupervised learning there are the obvious drawbacks of the fact that they usually take longer to learn and are not quite as accurate most of the time, but they can be so pliable. They can be used for such a large range of applications that when text summarization is used in the real world, this is typically the favorite. When we look at the difference between abstraction and extraction based summarizing it becomes a much more difficult issue to analyze. The outcome in the outside world is that a majority of the time and energy that goes into researching this topic, goes into extraction-based summarizing. This is true for many reasons, the biggest of which is that it is just much easier to implement and to get working successfully. But extraction-based summarization has a ceiling and that is, it does not matter how good it is at selecting important sentences, it will always be a cut and pasted summary of sentences that were torn out of their context and put in a new context. Even the best of these summaries are choppy because of this. Abstraction based summarizers are much more difficult to understand and to implement successfully, and they are harder to get good numbers for, but the theory behind it allows for near complete sentence regeneration in the summary of these documents. This characteristic alone sets it apart in a part of the field that could revolutionize the entire field as a whole. This aspect of abstraction based summarizing gives it such great potential that, “automatic abstractive summarization is the ultimate goal in this field”(source 2b). The issue is, all of the research is going into extraction based summarizers because they do the job that it is asked, and even though using an abstraction based learner is a much better display or artificial intelligence and it is much more beautiful to see come together, extraction based summaries give the user what they want, the most important sentences. The preference towards extraction-based summarizers is seen when we take a look into the direct applications for today. These applications are of a wide variety including, news reporting, giving summaries on a business’s history, giving summaries for doctors on a doctor’s notes, or a summary of a law report. According to source 5 the places that this field could have the most growth in in the future is in search engines, and document visualization.

I have seven different sources pertaining to this topic of text summarization. All of the sources are listed with the links to their location at the bottom of this paper, and they are titled with the purpose for which they have been included in the paper. Source 1 is an in depth look into an implementation of Naive Bayes in order to make a Extraction-based summarizer. This is where I got almost all of my information contained in the first body paragraph. This algorithm was full of equations that were not adequately defined in the source and so this resource took a lot to try to understand. The source 1b is a small source just to define Term Frequency and Inverse Document Frequency because I was having a difficult time finding a clear definition in the source. Project 2 was the implementation of the TextRank algorithm. This is where I got the majority of my information for what I described in the second body paragraph. This source had implementations for TextRank for finding the important sentences in a document and one for finding the keywords. I chose to implement to one looking for the important sentences because the implementation for keywords was a supervised learner as well and just very similar to the algorithm I talked about in the first body paragraph so including it I thought would be redundant. The source 2b I got because I thought it would be useful for explaining the second algorithm but it turned out to not be that useful at all for that purpose, but it had a lot of good information on the applications of Text Summarization and the future of it as well. The source 3 was a document that detailed the third algorithm I used. For this algorithm I had particular difficulties getting a document that was useful because it was not used to implement in abstraction-based summarizers because most abstraction-based summarizers are unsupervised learners. This document was very hard to follow as well and the concept was very confusing. I did stop before the document was finished with the algorithm, but I stopped where I determined that it had described to formula to the extent that I needed. The 4th source was what I used to describe the 4th implementation of an algorithm. This was where I got almost all of my information for the 4th body paragraph and I did leave out some technical add-ons towards the back that I thought were not necessary to understanding abstraction-based summarization. Source 5 was a book on text summarization and mainly the only parts I used was just for some general knowledge of abstraction vs extraction and which is used more and things of that nature.

Works Cited

Source 1, algorithm 1: <https://www.hindawi.com/journals/tswj/2016/1784827/>

Source 1b, algorithm 1: <http://www.tfidf.com/>

Source 2, algorithm 2: <https://web.archive.org/web/20120617170501/http://acl.ldc.upenn.edu/acl2004/emnlp/pdf/Mihalcea.pdf>

Source 2b, algorithm 2: <https://repository.asu.edu/attachments/158119/content/Nadella_asu_0010N_15317.pdf>

Source 3 algorithm 3: <http://blog.datumbox.com/machine-learning-tutorial-the-max-entropy-text-classifier/?utm_content=buffer15a8c&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer>

Source 4 algorithm 4: <https://arxiv.org/pdf/1506.01597.pdf>

Source 5 text summarization in general: <http://www.imperial.ac.uk/pls/portallive/docs/1/18619759.PDF>