CoNFET: AN ENGLISH SENTENCE TO EMOJIS TRANSLATION ALGORITHM

Alex Day[1] , Chris Mankos [1] , Soo Kim[1] , Jody Strausser[1]

[1]Clarion University of Pennsylvania, Computer Information Science Department

{A.D.Day C.F.Mankos}@eagle.clarion.edu, {skim, jstrausser}@clarion.edu

ABSTRACT

Emojis are a collection of emoticons that have been standardized by the Unicode Consortium. Currently, there are over 3,000 emojis in the Unicode standard. These small pictographs can represent an object as vague as a laughter (🤣) to something as specific as a passport control (🛂). Due to their high information density and the sheer amount, emojis have become prevalent in common communication media such as SMS and Twitter. There is a need to increase natural language understanding in the emoji domain. To this end, we present the CoNFET (Composition of N-grams for Emoji Translation) algorithm to translate an English sentence into a sequence of emojis. This translation algorithm consists of three main parts: the n-gram sequence generation, the n-gram to emoji translation, and the translation scoring. First, the input sentence is split into its constituent n-grams either in an exhaustive manner or using dependency relations. Second, the n-grams of the sentence are translated into emojis using the nearest neighbor in a vectorized linguistic space. Finally, these translations are scored using either a simple average or an average weighted by the Term Frequency-Inverse Document Frequency (TF-IDF) score of the n-gram. As the result, the sequence of emojis with the highest score is selected as an output of the sentence summarization.

KEY WORDS

Natural Language Processing, Abstractive Text Summarization, Emoji, Word2Vec, Word Embeddings, TF-IDF

1 Introduction

Automatic text summarization is a category of algorithms that aims to produce a small set of representative information from a larger input document. There are two general categories of automatic summarization: extractive and abstractive. Extractive summarization *extracts* sentences and phrases that are already present within the document in order to produce an outline. While this method can produce representative summaries, it does have the drawback of limiting the vocabulary that can be used in the summary to the vocabulary within the original document. Conversely, abstractive summarization tries to understand the information within the document and summarize it by creating new phrases and sentences. In our previous research [1], we implemented two extractive summarization methods: Term Frequency-Inverse Document Frequency (TF-IDF) and TextRank and presented the results of running the algorithms on three different corpora: *Moby-Dick* by Herman Melville, a selection of Reuters news articles, and a selection of posts on Reddit. Both algorithms produced representative summaries from the short stories from Reddit. However, they did not perform optimally when presented with larger works which possessed a more complex story arc.

In the domain of Natural Language Processing (NLP), word embeddings [2] allow machines to produce representative, fixed-length vectors from a single word. This was a massive breakthrough in NLP mainly because many machine learning algorithms take a fixed-length vector as their input. Most previous attempts of word embeddings were lossy or produced vectors of an unwieldy length. Another benefit of this vector representation is that it allows a direct numerical comparison between the words. This work has been expanded from words to both emojis [3] and sentences [4].

Our research aims to tackle abstractive text summarization in a novel way by compressing an English sentence into a series of emojis. Emojis are a pictographic language that is commonly used on the Internet and within text messages. In the latest emoji standard, there are 2,823 individual characters that can be both used alone or conjoined with modifiers to change, among other things, skin tones and gender. Because of their information density and lack of formal grammar emojis are ideal for text summarization. Emoji denseness allows the compression large chunks of the sentence into a single emoji. For example, the running emoji (🏃) can represent a man who is running away from something or a man participating in a track event in just one character. The lack of formal grammar avoids a large issue present in most neural machine translation, so we can focus on the adequacy rather than the fluidity of the produced translation.

There are two primary uses for our algorithm: text summarization and communication facilitation. Producing a summary of a document using emojis could prove to be a quick way to let someone decide if a document is interesting to her/him without reading the whole document. It could also be used to facilitate communication across language barriers between people with learning disabilities [5].

2 Related Work

The algorithm presented in this paper is based on the concept of embeddings [6]. Embeddings are a way of representing human text in a machine-understandable format. Normally this representation is a vector of floats with a range from -1 to 1 and a length of 300 to 700 depending on the application. The specific sentence vectorization model that our algorithm uses is Sent2Vec [4]. This model is based on Word2Vec [2] and extends the word vectorization into a representative sentence vectorization. Emoji2Vec [3] was also developed from Word2Vec, so that it can embed emojis in the same semantic space. This allows a direct vector comparison between words and emojis.

There have been other systems that accomplish a similar task in different ways. Emoji Dick [7] is a crowd-sourced translation of Moby Dick into emojis using Amazon Mechanical Turk crowd-sourcing platform. In addition to explicit translation, some email clients are using pictographs to augment communication for individuals with cognitive disabilities [5]. Outside of the realm of emojis for text translation, there is also research in the area of caption generation for images using emojis [8; 9]

The n-gram is a common motif within the domain of NLP. An n-gram is a sequence of n contiguous terms within a document. The term can be anything like characters, syllables, or words. However, an n-gram is composed of words in our application. Commonly, unigrams, bigrams, and trigrams are used, which are n-grams of length one, two, and three, respectively. N-grams are pertinent in their use as the terms within the Term Frequency-Inverse Document Frequency scoring and as the units of comparison for a given sentence when evaluating the cosine distances. Both of these topics are discussed in Section 3.3. To better understand our scoring methods, we will briefly discuss the concepts of Term Frequency and Inverse Document Frequency in this section.

Term Frequency (TF) [10] is how often a given term appears in the document being summarized. While there are multiple ways to calculate the TF value, it is typically defined by Equation (1):

$$TF(t,d) = f_{t,d} \qquad (1)$$

where f is the number of occurrences, t is the term, and d is the document.

Inverse Document Frequency (IDF) is the relation of how many documents are in the corpus against how many of those documents contain a given term. IDF is often defined as the logarithm of how many documents are in the corpus divided by how many documents the given term appears in as shown

in Equation (2):

$$IDF(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \qquad (2)$$

where t is the term, d is the document within the corpus, and D is the corpus.

Term Frequency-Inverse Document Frequency (TF-IDF) is the product of Term Frequency (TF) and Inverse Document Frequency (IDF) as shown in Equation (3).

$$TF - IDF(t,d,D) = TF(t,d) \cdot IDF(t,D) \qquad (3)$$

where t is the term, d is the document, and D is the corpus.

Common words, such as "the," will appear in many documents and have a lower TF-IDF score. Words pertinent to the document should be relatively frequent within the document and less frequent in the corpus. As the result, the relevant words will boost its TF-IDF score. For instance, imagine a corpus of three sentences: "The dog walks quickly.", "An apple falls from the tree.", and "The hero acted quickly." The Term Frequency of "the" within the first sentence is $\frac{1}{4}$ since 25 percent of the words are "the." However, there are three documents and "the" is in every document. Thus, its Inverse Document Frequency is $\log(\frac{3}{3}) = 0$ and the TF-IDF score for "the" is $\frac{1}{4} \times 0$. On the other hand, the Term Frequency of "dog" within the first sentence is $\frac{1}{4}$. However, since it is only in the first sentence, its Inverse Document Frequency is $\log(\frac{3}{1})$ and the TF-IDF score for "dog" is $\frac{1}{4} \times \log(\frac{3}{1})$, approximately .275.

3 Emoji Translation Algorithm

The CoNFET (Composition of N-grams for Emoji Translation) algorithm developed contains three disparate parts. These three modules can be easily swapped in and out to change the functionality of a specific procedure. The three parts of the algorithm are the n-gram sequence generation, the n-gram to emoji translation, and the translation scoring. Each of these topics are discussed in further detail in Sections 3.1, 3.2, and 3.3, respectively. The overall flow of the system is described in Figure 1. In this example, the input sentence "abc" is first split into n-gram sequences. Each sequence is translated into representative emojis during the second step. Each emoji sequence is then scored using one of the scoring metrics such as a simple average and an average weighted by the TF-IDF score. The emoji sequence with the highest score is returned as the summary.

3.1 N-Gram Sequence Generation

The first part of the CoNFET algorithm is splitting the input sentence into a series of n-grams denoted as an n-gram se-
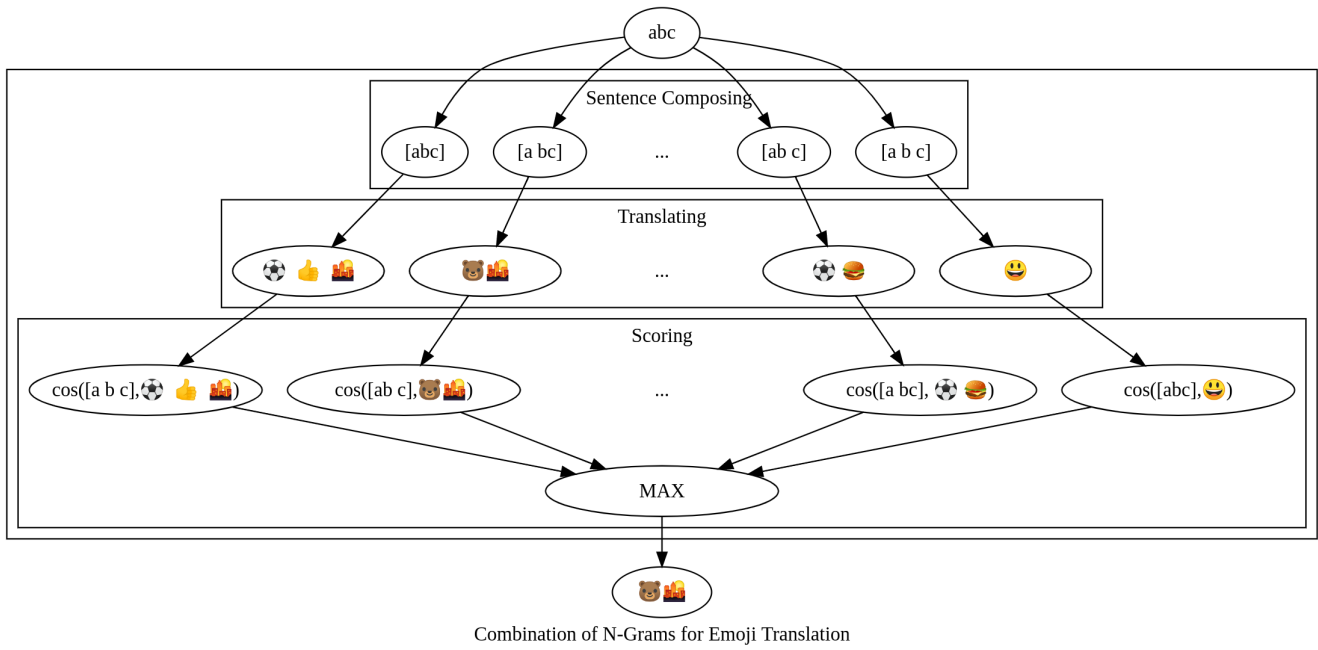
Combination of N-Grams for Emoji Translation

Figure 1: The overall flow of the Combination of N-grams in Emoji Translation (CoNFET) system

quence. For example, the unigrams from the sentence "The dog bit me" would be "the," "dog," "bit," and "me." Similarly, the bigrams from that same sentence would be "The dog", "dog bit", and "bit me". This sequence of n-grams would ideally split the sentence into its constituent words and then translate them into emojis. Two approaches have been applied to generate n-gram sequences: exhaustive n-gram generation and dependency tree informed n-gram generation. Section 3.1.1 describes exhaustive generation, meaning that every n-gram sequence is generated and tested. Section 3.1.2 describes the second approach that generates the sequence in a way informed by the relations between words in the input sentence.

3.1.1 Exhaustive N-Gram Sequence Generation

The initial approach for the n-gram sequence generation was a brute force approach that rests on the assumption that the optimal n-gram sequence is made up of contiguous words within the input sentence. That is, if the input is of the form "a b c," the optimal n-gram sequence will never contain "c a," "c b," nor "b a." The optimal n-gram sequence is defined as the sequence that produces the best summary. These n-gram sequences are generated by producing all compositions of the integer that is equal to the length of the sentence. Each of those compositions is then extrapolated into an n-gram sequence. For example, the integer 3 can be generated by summing the elements in the following arrays: [1, 1, 1], [1, 2], [2, 1], or [3]. Any one of those arrays can be turned into an n-gram sequence by taking n-grams that are equal to the length of the element from the input sentence. For example,

[1, 1, 1] is translated into a sequence of 3 uni-grams. The major downside to this approach is computational complexity. For a sentence with the word length n, this algorithm generates 2^{n-1} sequences. While speed was not a major concern with this algorithm, sentences with 10 or more words take prohibitively long to translate, as shown in Figure 2.
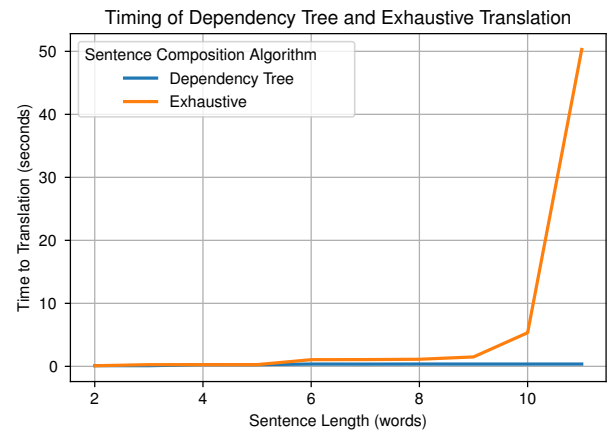

Figure 2: Translation time of two n-gram sequence generation algorithms

Figure 2 depicts the translation time for sentences with different lengths using the exhaustive and dependency tree sentence composition algorithms. If the input sentence has more than

10 words, the translation time using the exhaustive algorithm increases exponentially. In fact, sentences with more than 11 words results in a memory exception in the development environment when trying to summarize with the exhaustive algorithm. Note that the CoNFET algorithm was executed through a Jupyter Notebook running within a Docker container on a system exposing one core of an Intel Xeon 6140 and 16 GB of RAM to the virtualized system.

3.1.2 Dependency Tree Informed N-Gram Sequence Generation

The second approach that was used to generate a representative n-gram sequence is the dependency relations within a sentence. Figure 3 shows an example of the sentence "I finished my homework just before class started." with the part of speech tagging and the dependency relations. The dependencies can be thought of as modifiers in the sense that the pronoun "I" modifies the verb "finished" to inform who executed the action.
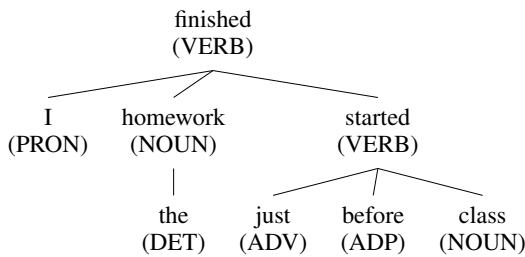
Figure 3: Dependency tree for "I finished the homework just before class started."

The tree in Figure 3 has certain redundancies that can be removed in order to form larger groupings of words as nodes. There are two rules used for this tree collapse:

(1) Parent and child relationship: If a parent has only one child, then the parent and the child are combined.

(2) Leaf relationship: If two or more leaves are on the same level with the same parent, then these leaves are combined.

An example of the first tree collapse rule, that is, the parent and child relationship collapse, can be seen in Figure 4 and Figure 5. The nodes "homework" and "the" are highlighted in Figure 4 to indicate that they fulfill the requirement of a parent with only one child. These two nodes can be collapsed into one node as shown in Figure 5.
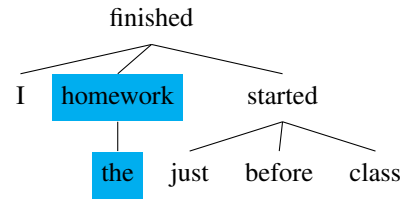
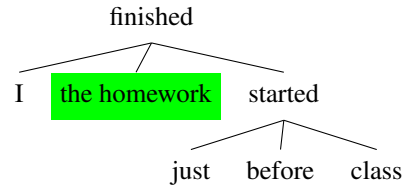Figure 4: Before the parent and child relationship tree collapse

Figure 5: After the parent and child relationship tree collapse

An example of the second tree collapse rule, that is, the leaf relationship collapse, can be seen in Figure 6 and Figure 7. In Figure 6, the three leaf nodes "just", "before", and "class" are highlighted to indicate that they are the children of "started" and they are on the same level. These three nodes can be collapsed into one node as shown in Figure 7.
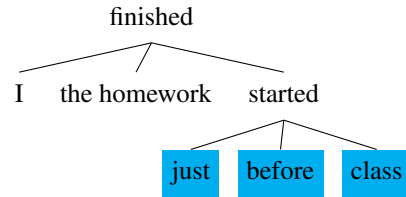
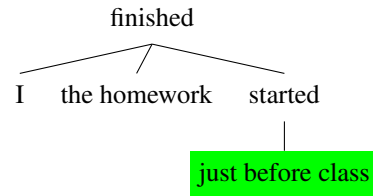Figure 6: Before the leaf relationship tree collapse

Figure 7: After the leaf relationship tree collapse

3.2 N-Gram to Emoji Translation

The next step in the CoNFET algorithm is to translate an n-gram sequence into an emoji sequence. At its most basic, an emoji is purely a visualization of a keyword. For example,

the dog emoji (🐕) is a visual representation of any of the following: "dog," "puppy," "Shiba," etc. With this in mind, it is now simpler to find a way to draw comparisons between n-grams and emojis. Both the emoji description and the n-gram in question can be vectorized, so that a direct mathematical comparison can be made. The vectorization is performed using Sent2Vec [4]. Once the emoji and the n-gram are in the same format, the cosine distance calculation as shown in Equation 4 is used for a direct similarity comparison. The best emoji is defined as the emoji with a description that has the lowest cosine distance to the n-gram.

$$cosine\ distance = 1 - \frac{\vec{u} \cdot \vec{v}}{||\vec{u}||_2 ||\vec{v}||_2} \tag{4}$$

where $\vec{u}$ and $\vec{v}$ are the two equal-length vectors that the cosine distance is being calculated for.

3.3 Translation Scoring

Each n-gram sequence generated in Section 3.1 is given a score based on the individual n-gram cosine distance. Then, a sequence of the emojis corresponding to the highest score becomes the output of the summarization for the input sentence. Average cosine distance scoring and weighing metric using TF-IDF methods are used in our algorithm and discussed in Section 3.3.1 and Section 3.3.2, respectively.

3.3.1 Average Cosine Distance Scoring

A score for the overall n-gram sequence is generated by adding up the cosine similarity of each n-gram composing the sequence and then dividing by the number of n-grams as shown in Equation 5. The average cosine distance was chosen as a metric because it is computationally inexpensive to calculate and it allows all n-gram scores to contribute equally into the overall score of the translation.

$$Average\ Cos\ Distance\ Score = \frac{\sum_{i=1}^{k} s_i}{k} \tag{5}$$

where s_i is the cosine similarity for the ith n-gram in the given n-gram sequence and k is the total number of n-grams within the n-gram sequence.

3.3.2 Weighing Metric using the TF-IDF

There are words that do not carry much semantic weight with extraordinarily small cosine distances to certain emojis. For instance, the word "a" maps directly to 💯. In order to address the varying impact of individual words, inspiration is drawn from one of the abstractive summarization techniques, Term Frequency-Inverse Document Frequency (TF-IDF).

Using an open-source Python library Gensim [11], the TF-IDF scores are created for every term in a given corpus. It is impractical to create n-grams of every possible length as the input sentence could be any length; therefore, the document frequencies for all n-grams except unigrams are estimated. To generate such a document frequency for a given n-gram, two estimates are needed: the probability that each term within the n-gram occurs together in a document and the probability that the terms occur in the proper order. A naive estimate of the chances of the constituent terms occurring in the same document can be obtained by multiplying their document frequencies together as shown in Equation 6.

$$P(a_1 \wedge a_2 \wedge \ldots \wedge a_k \text{ in } d \in D) = \prod_{i=1}^{k} \text{DocFreq}(a_i, D) \tag{6}$$

where a_i is the ith term in the n-gram, k is the total number of n-grams within the n-gram sequence, d is the document, and D is the corpus.

In order to obtain a similarly naive estimate of the terms occurring in the necessary order, the input sentence is used as a template. Considering the number of rearrangements of the words in the input sentence with the terms in the n-gram in the proper order against all possible rearrangements of the words gives the probability that the n-gram appears in that sentence. Multiplying both of these values together gives a weight for each individual n-gram within the sequence. While there are issues with treating the words as independent and with treating a sentence as representative of a document, this allows us to handle arbitrary n-grams as well as keep the dictionary in a manageable size. There exist words like "New" and "York" that are frequently paired and in some corpora occur together more often than either word independently. Now having a score for each n-gram within the sequence, a score for that particular sequence can be obtained by weighing each n-gram cosine similarity with its TF-IDF score and then averaging by the TF-IDF score as shown in Equation 7.

$$Average\ score\ weighted\ by\ \text{TF-IDF} = \frac{\sum_{i=1}^{k} w_i \times s_i}{\sum_{i=1}^{k} w_i} \tag{7}$$

where s_i is the cosine similarity score of the ith n-gram and w_i is the corresponding TF-IDF weight.

3.4 Sentiment Emoji

One complication with the summarization process is that multiple phrases map to the same emoji. With the few human translations considered, the emotion behind the input sentence was not obvious after translated into emojis. For example, both sentences "Happy dog treats dog" and "Sad dog treats dog" map to 🐕🍖🐕. In an attempt to facilitate human translation, a sentiment emoji was proposed. Tagging the sentence with a 😊 or a 🙁 might allow humans to get

closer to the input. Using the Python library TextBlob [12], a sentiment score is obtained for the input sentence. It results in the polarity and subjectivity scores that are corresponding to how positive and how factual TextBlob ranked the input, respectively. Initially, the table in Figure 8 was used and the polarity is ranged from -1 to 1 and the subjectivity is ranged from 0 to 1.


Figure 8: Table used in conjunction with TextBlob for initial sentiment analysis

In order to allow more than the nine emojis seen in Figure 8, and use emojis more in line with how they are used on social media, emojis were scored using tweets. Using Twitter's API, between 30 and 100 tweets were gathered for each emoji. Each tweet was scored using TextBlob and the scores for each emoji were averaged. Then, the emoji with the average sentiment score closest — using the standard deviation as the distance — to the sentiment score of the input sentence is selected as the sentiment emoji. The results from scoring tweets are shown in Figure 9. Many emojis end up with the polarity and subjectivity values close to zero, which does not allow distinguishing between emojis when considering distance from the input. There are two reasons for this: the size of the dataset and the lack of preprocessing of the tweet data. First, due to Twitter's limitation on daily API queries, the dataset gathered was fairly limited. This issue is easily addressed by gathering data for a longer time. Second, misspellings, slang, and grammatically incorrect sentences yield poor results in TextBlob. For example, even a simple misspelling such as "tightt" for "tight" is enough to give the tweet a score of zero. This preprocessing issue could be fixed by including a more complex cleaning pipeline before sentiment analysis. An implementation linking an existing public emoji-sentiment dataset from something like Kaggle to sentiment scores of the input sentence instead of scoring the tweets using TextBlob might produce better results. This consideration was initially discarded for two reasons: first, because the largest dataset found only had around 1000 emojis and approximately 80 percent of emojis had fewer than 100 occurrences. Second, the emojis are tagged with their number of positive, negative, and neutral occurrences. How to appropriately map this onto TextBlob's two dimensional subjectivity/polarity scores was not immediately obvious. However, there are far fewer emojis that would be clustered at zero.

4 Test Plan

Translation algorithms are commonly evaluated by comparing the output from the algorithm to the known good translations. However, because there has not been much research done in this English to emoji translation domain, there was not a large enough amount of sample data to test this algorithm in that fashion. Furthermore, the allocated research time did not allow for the creation of such a data set. Due to this fact, a new way to quantify the accuracy of the translation algorithm was needed. The testing algorithm developed starts by generating an emoji translation for a given number of input sentences. These emoji translations are presented to a user for translation back into English sentences. A direct numerical comparison can then be drawn between the input sentence to the algorithm and the sentence provided by the human using the cosine distance between the vectorized sentences. One major downside to this testing is that it is human-in-the-loop, meaning that human interaction is needed to complete the test. This leads to a slower overall testing process, introduces potential bias due to educational, cultural, and societal differences, and removes the ability to test at each iteration. The general flow for this test process is as follows:

(1) Generate sentences in English.
(2) Summarize each of the sentences using CoNFET.
(3) Take the top 20 sentences sorted by the certainty score.
(4) For each machine translated sentence:
    (a) Provide the user with the emojis.
    (b) Provide the user with the length of the result sentence.
    (c) Prompt the user to translate the emojis into a sentence.
(5) For each pair of the machine translated sentence and the user translated sentence:
    (a) Calculate the distance between the two sentences using Sent2Vec.

5 Implementation and Results

All of the code for this project was written in Python 3, using the following packages: SpaCy [13], NLTK [14], Gensim [11], TextBlob [12], Sent2Vec [4], NumPy [15], and Jupyter [16]. The dataset used is the dataset from the emoji2vec [3].

The results from the exhaustive n-gram generation algorithm are presented in Figure 10. The results include the sentence inputted to the algorithm, the emojis generated from the algorithm, and the score attributed to each translation. Here, the average cosine distance was used. Because emojis are just visualizations of specific descriptions as mentioned in Section 3.2. As such, some of these translations may not make sense unless the specific descriptions of emojis are known. For example, the n-gram "They are playing" is translated to the emoji 🎴 as shown in the third test data in Figure 10. This

Emoji	Polarity Mean	Polarity Std. Dev.	Subjectivity Mean	Subjectivity Std. Dev
😃	0.051	0.276	0.276	0.276
😁	-0.049	0.302	0.302	0.302
😂	0.008	0.146	0.146	0.146
😄	0.129	0.409	0.409	0.409
😆	0.081	0.199	0.199	0.199
😅	0.011	0.304	0.304	0.304
😆	0.16	0.215	0.215	0.215
😇	0.265	0.432	0.432	0.432
😈	0.064	0.161	0.161	0.161
😉	0.125	0.2	0.2	0.2
😊	0.387	0.48	0.48	0.48
😋	0.212	0.331	0.331	0.331
😌	0.08	0.271	0.271	0.271
😍	0.345	0.224	0.224	0.224
😎	-0.004	0.3	0.3	0.3
😏	0.115	0.54	0.54	0.54
😐	0.08	0.133	0.133	0.133

Figure 9: Preliminary results from scoring tweets

emoji is known as the "Flower *Playing* Cards." Once this description is known, the emoji translated sentence makes more sense.

Input Sentence	Output Emojis	Score
The dog runs fast	🍽️🏃🐶	0.984
The child was in love with the cat	👶😺🐾	0.824
They are playing christmas music from the bell tower	🚩🎄🎷⛪🏯	0.893
I think that this computer has a virus	💭💾🔌	0.769
I have to wear my headphones to run in the race	👨‍🎧🏃🏁	0.960
The company Apple makes both cell phones and computers	🍏📱💾	0.903

Figure 10: Results from the exhaustive n-gram generation algorithm

the dependency tree algorithm does not take into account the score that the translation of the proposed sequence will have. Alternatively, the exhaustive algorithm splits the sentence as to maximize the resulting score.

Input Sentence	Output Emojis	Score
The dog runs fast	🏃🍽️	0.663
The child was in love with the cat	👶😈🐾	0.629
They are playing christmas music from the bell tower	😈🚩🎄🏯	0.706
I think that this computer has a virus	👤⭕💯💾🔌	0.822
I have to wear my headphones to run in the race	👤🎿😈👤🎧🏃🎽	0.668
The company Apple makes both cell phones and computers	🍏😈📞📱	0.590

Figure 11: Results from the dependency tree n-gram generation algorithm

The results from the dependency tree n-gram generation algorithm are presented in Figure 11. Again, average cosine distance was used. These summary results are not as good as the results of the exhaustive algorithm. This is mainly because

6 Conclusion

In this paper, a new algorithm for translating an English sentence into a series of emojis, CoNFET is presented. Emojis

are becoming well understood in the domain of Natural Language Processing, so they make sense as an output medium for a new translation algorithm. CoNFET accomplishes this goal through the use of the n-gram sequence generation algorithms combined with the n-gram to emoji translation and the scoring methods. This algorithm could be useful for the field of communication between individuals with cognitive disabilities or for describing a long document in a condensed way.

One of the main drawbacks of this implementation was the limited dataset [3]. The deadest defines the complexity of the output language and it is used as a translation mapping from emojis to English phrases and vice versa. In the future, some exploration will be done with scraping different emoji database-like sites (e.g. Emojipedia) to create a richer output language. In addition to the size limitations, the dataset is also limited in the sense that one keyword may have multiple emojis that represent it. In fact, in our dataset, 15%, around 720 descriptions have multiple emojis. This means that some n-grams have multiple emojis that are all equally close. In the future, we will investigate alternate heuristics by using the other keywords of an emoji when deciding what translation is most suitable. Finally, the last problem is that each n-gram is translated individually. Therefore, no previous context in the sentence is used in the n-gram translation. If the previous n-grams can be taken into account while translating the current n-gram, this may produce a more representative emoji sequence. Also, the test plan detailed in Section 4 should be implemented and evaluated. While additional work in this area does need to be conducted to improve the algorithm, this initial effort has provided a meaningful start.

References

[1] A. Day, S. Kim, A Comparison Of Automatic Extractive Text Summarization Techniques, in *34th Annual Spring Conference of the Pennsylvania Computer and Information Science Educators (PACISE)*, pages 98, 102 (PACISE, 2019).

[2] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781*.

[3] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bosnjak, S. Riedel, emoji2vec: Learning Emoji Representations from their Description, in *Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media* (Association for Computational Linguistics, 2016), doi: 10.18653/v1/w16-6208.

[4] M. Pagliardini, P. Gupta, M. Jaggi, Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features, in *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics* (2018).

[5] V. Vandeghinste, I. S. L. Sevens, F. Van Eynde, Translating text into pictographs, *Natural Language Engineering*, *23*(2):(2017), 217–244.

[6] J. Brownlee, What are word embeddings for text?, `https://machinelearningmastery.com/what-are-word-embeddings/`, 2017, (Accessed on 02/11/2020).

[7] W. Radford, B. Hachey, B. Han, A. Chisholm, : telephone:: person:: sailboat:: whale:: okhand:; or "Call me Ishmael"–How do you translate emoji?, in *Proceedings of the Australasian Language Technology Association Workshop 2016*, pages 150–154 (2016).

[8] B. Mazoure, T. Doan, S. Ray, EmojiGAN: learning emojis distributions with a generative model, in *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 273–279 (Association for Computational Linguistics, Brussels, Belgium, 2018), doi:10.18653/v1/W18-6240.

[9] S. Cappallo, T. Mensink, C. G. Snoek, Image2emoji: Zero-shot emoji prediction for visual media, in *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1311–1314 (ACM, 2015).

[10] J. Leskovec, A. Rajaraman, J. D. Ullman, Data Mining, in *Mining of Massive Datasets*, pages 1–18 (Cambridge University Press), doi:10.1017/cbo9781139924801.002.

[11] R. Řehůřek, P. Sojka, Software Framework for Topic Modelling with Large Corpora, pages 45–50 (2010), doi:10.13140/2.1.2393.1847.

[12] S. Loria, TextBlob, `https://github.com/sloria/TextBlob`, 2019.

[13] M. Honnibal, I. Montani, spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing, 2017, to appear.

[14] S. Bird, E. Klein, E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit* (O'Reilly Media, Inc., 2009).

[15] T. Oliphant, NumPy: A guide to NumPy, USA: Trelgol Publishing, 2006–, [Online; accessed ¡today¿].

[16] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, Jupyter Notebooks – a publishing format for reproducible computational workflows, in F. Loizides, B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90 (IOS Press, 2016).