Text-Based Animated GIFs Recommender for Twitter using Natural Language Processing

Yiran Cao, Guanzhao Wang, Haochen Wu
{y267cao, g95wang, h286wu}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Introduction

The objective of this paper is to compare, discuss, and develop a text-based animated GIFs Recommender that can be incorporated into many social media platforms such as Twitter, Messenger, WeChat, WhatsApp, etc. Using Natural Language Processing, we can extract keywords from text messages, then generate a list of recommended GIFs that are suitable for the context. **We will use a Recurrent Neural Network (RNN) model with a Bidirectional-Long-Short-Term-Memory (BLSTM) layer to extract keywords from tweets, and use a statistical approach named Term Frequency-Inverse Document Frequency (TF-IDF) as a baseline method to be compared with our method.**

Comparing to plain text or still images, GIFs are dynamic, which allows the users to express their messages and emotions more easily in a simple animation. GIFs also make the connotation of messages more explicitly, reducing the chances of the misunderstanding of messages. Additionally, GIFs can lighten the mood of social media platforms, which may also enhance the emotional bonds among people.

Also, compare to videos, GIFs are of much smaller size, and it is simpler to create, use, and spread in social media applications. There are plenty of websites for online GIFs creation, and users can create their GIFs in one minute. Moreover, users can forward GIFs sent from their friends with one single click. As a result, GIFs can be rapidly broadcast to other users.

GIFs have been an essential part of social media platforms. Google says it now sees "millions of searches for GIFs every day," and Tenor, a GIF search engine affiliated to Google, is now as popular as Twitter. Nearly 300 million users performs over 12 billion GIF searches every month on its site (Tenor 2018). Since GIFs are making an enormous influence on the way that people communicate nowadays, this recommendation system would be an intriguing feature to be incorporated into these social media applications to further enhance user experience.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The problems that we are primarily attacking is keyword extraction from text messages, in particular, tweets, using Natural Language Processing. The basic workflow is to extract the core information or the keywords from it. Then, we use the results from the above to perform a search in an animated GIF library.

The primary focus is the two training algorithms to be implemented. In this paper, we will compare the performance of semi-supervised deep learning with RNN and BLSTM and unsupervised statistical methods called TF-IDF for keyword extraction by computing the F-1 score of the two methods evaluated with existing labeled data-set from the paper "*Automatic Keyword Extraction on Twitter*" (Marujo et al. 2015). For the RNN approach, we design a neural network for the training inputs, and since we do not have sufficient labeled data, we perform semi-supervised training. The TF-IDF approach applies statistical computation to produce a confidence probability for each word as being a keyword of the text. We use the probability to find keywords in the text. To improve the performance, we annotate Twitter data with POS and dependency taggings as extra features for our neural network.

Related Work

As a branch of Artificial Intelligence, Natural Language Processing (NLP) focuses on processing and analyzing human language. It is one of the earliest areas of interest in the field of Artificial Intelligence when most research has been devoted to the application of language translation.

Back in 1950, Alan Turing proposed the Turing test as a benchmark of intelligence, which examine how well a computer program can imitate a human being, including thinking and acting in a style that a human cannot distinguish between the program and a real human based on the conversational content alone. (TURING 1950) Notice that for the computer to behave like a human, the ability to understand and speak a human language is inevitable, and hence, NLP is an essential part of machine intelligence in the Turing test.

In the 1960s, some notable progress is made in this area, including SHRDLU (Winograd 1971) and ELIZA (Weizenbaum 1966), both of which are primitive NLP computer programs having the ability to interact with a human using natural languages. Later on, an early example of a chatterbot PARRY was invented (Colby 1974). In the meantime, many

other NLP programs are developed, all of which are considered Symbolic NLP. The symbolic path along the development of NLP is a long one and is one of the first attempts to tackle this problem, where words are considered no more than mere tokens without any semantic meanings, and sentences are treated as an order-less collection of tokens. To understand a sentence, the program would extract from the tokens the keywords that the programmer has hard-coded the meaning or the possible actions that the program can perform. Similarly, the program would collect some necessary words and assemble them into a human-readable sentence using some grammatical rules to produce outputs.

Up to the 1980s, most NLP programs adopt the symbolic methodology, requiring a large but never a sufficient amount of complex hand-written grammatical rules. Alongside the development of hardware to increase the computational power and the revolution in linguistics where the dominance of Chomskyan theories (Chomsky 1986), the ones addressing the language acquisition as part of human instinct, had been lessened, exploration of other methods for NLP has been conducted. One of the most successful approaches is statistical NLP, where unsupervised and semi-supervised learning algorithms are introduced in the context of NLP. Such algorithms can learn from data independently using statistical inferences without the requirement of the usage of enormous rules provided by a human. Decision trees, as one of such examples of machine learning algorithms, produced rules similar to existing hand-written ones (Orphanos et al. 1999).

Around early 2010s, researchers started to adapt neural networks in machine learning to enable the automatic training procedure, instead of performing statistical methods only. The invention of artificial neural networks extremely revolutionized the field of AI, including NLP.

The keyword extraction problem has been intensively studied in the NLP field. In general, two approaches have been used to address this problem: the unsupervised approach and supervised approach.

Keyword extraction is considered as a ranking problem for unsupervised approach. Term Frequency Inverse Document Frequency (TF-IDF) was a widely-used measure for this problem (Jones 1972). The tf-idf value is result of the multiplication between term frequency and inverse document frequency. It is shown that TF-IDF works well when extracting keywords from articles and documents. (Kaur and Gupta 2010)

Liu et al. approached the keyword extraction problem by using a clustering method. All candidate terms are first selected and grouped into clusters, and the exemplar terms are identified for each cluster. Then key phrases are extracted from the document based on the exemplar terms (Liu et al. 2009).

At the same time, lots of supervised machine learning models have also been developed to tackle the keyword extraction problem.

In 1999, a widely known model called KEA was developed by Eiber Frank. This model takes TF-IDF and the location of the first occurrence of the word into account, and computes the measure of keyword extraction problem with the Naive Bayes model. Candidate phrases are first generated and assigned with a higher priori. Then the TF-IDF and the first appearance of the word are used as the likelihood to compute the posterior probability in Bayesian Learning. The model will output a list of phrases ranked according to their probability of being a keyword (Wu et al. 2005).

Benuyed et.al solved the problem by using the confidence measure and the Support Vector Machine (SVM). To extract keyword using the SVM approach, the data first go through a recognition phase, where all the keywords are picked out. Then, a hidden Markov model (HMM) based speech recognizer is used to calculate the confidence measure of each word. Finally, SVM is applied for classifying a sequence of detected keywords into correct and incorrect keywords based on the confidence measure (Benayed et al. 2003).

There are also recent researches that focus specifically on extracting keywords from twitter. Comparing to formal articles and documents, twitter posts are very different for several reasons. First, twitter posts are limited to at most 140 words. Given a short paragraph and the keyword may only appear once in the text, many of the unsupervised approaches will produce unreliable results. Additionally, the language is also more casual with many messages containing orthographical errors, slang, and abbreviations that are domain specific, which makes text processing even harder (Marujo et al. 2015).

Zhang et. all, in 2016, after identifying all difficulties on keyword extraction on Twitter, proposed an RNN model that works specifically for Twitter posts. This model contains two hidden layers, where the first hidden layer is for keyword information retrieval, and the second layer is used to apply a sequence labeling method to extract the key phrases with regard to the keyword information. The proposed RNN has also shown to give a better result when dealing with Twitter posts than all other approaches at that time. (Zhang et al. 2016)

Methodology

Text processing is generally a complicated task. Previous work adopting the symbolic methodology which uses hand-written grammatical rules to process the texts. This requires a large amount of human-annotation, and does not scale well, so it is generally recognized as a dead-end nowadays. Instead, the invention of neural network revolutionized the field, which can predict the underlying function from the data-set with minimum supervision from human. Hence, we attempt to tackle the task with a neural network.

Recurrent Neural Networks (RNN) are a popular choice when it comes to natural language processing, because it is a type of network that takes sequential data as input. For example, a sentence is an ordered sequence of words. Different rearrangement of the words may lead to drastic changes of the meaning. Therefore, an RNN model which can incorporate this ordering would be our first choice if we decide to use a neural network.

We implement an RNN model to perform the keyword extraction task based on research by Zhu, et.al (Zhu et al. 2020). The structure of the model is obtained fromthe above

paper, and is demonstrated in Figure 1. To train the network, we need more information other than the text itself to achieve a desirable performance. As an example, articles and pronouns may repeat frequently in the text, and can be recognized as keywords if we feed only the plain text into the network. To avoid this situation, we need more information to perform the training.

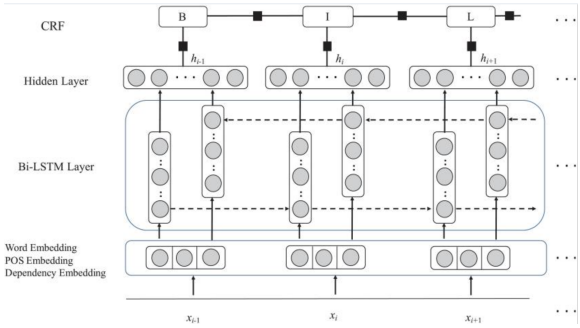

Figure 1: RNN Model

Part-of-Speech (POS) tagging and dependency parsing are two commonly used types of annotations in the literature. POS tagging labels each word in the text by adding a corresponding part of speech tag based on definition of the words and the context. These tags identify the words as noun, verbs, adjectives, etc. Dependency parsing, on the other hand, extracts the information regarding the grammatical rules. It can, for example, identify if a part of the sentence is a clause or a phrase.

Now that we have the annotated data-set, we can encode each word in the text as a feature along with its POS and dependency taggings. Each features are sequentially fed into a Bidirectional-Long-Short-Term-Memory (BLSTM) layer. The BLSTM layer takes in one feature each time, which in this case is a single encoded word annotated with POS and dependency taggings. The layer then processes the feature to generate some information or new features, and decides if any information is valuable, and pass some but not all of it to the next layer, some to remember, which is saved and reused when processing the next features, and some to discard. Those saved information are used as extra input features for the rest of the text. Notice that we feed the word features into the network sequentially rather than in parallel, in contrast to the methodology of multi-layer perceptrons. Since this layer is bi-directional, the features are passed into the layer twice, once in the forward direction and once in reversed order. In this way, the dependency of the text structure is well-preserved, since each word can gain information from words before and after that particular word.

Then, we use one hidden layer to summarize the output of the BLSTM layer from the two directions and use the summarized result to help produce the label of the text. We adapt the BILUO label scheme developed by Zhu, (Zhu et al. 2020). This label scheme labels each word of the text, distinguishing if this word is itself a keyword or part of the keyphrase of the entire text. We use Conditional Random Field to predict the labels using the features from BLSTM layer.

Conditional Random Field (CRF) is a statistical model formed based on probability, which is commonly used for modeling sequential data by incorporating the important features of the nearby data points. CRF uses a number of feature functions to evaluate the accuracy of the given label. In text processing, for example, the feature function $f(s, i, l_i, l_{i-1})$ in the linear-chain CRF takes four inputs, the sentence $s = \{w_1, w_2, \dots w_n\}$, the position of the current word in the sentence i, the label of the current word l_i, and the label of the previous word l_{i-1}, and produce a score of the assigned label for the current word, which also takes into account the information of the previous word. Then, summing up the weighted feature scores of all words appearing in the sentence, we derived a measurement of the sentence with respect to the specific label sequence l. (Zhu et al. 2020)

$$score(l \mid s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1})$$

Later on, we can find the conditional probability of this specific label sequence given the sentence, which is computed as follows:

$$p(l \mid s) = \frac{\exp[score(l \mid s)]}{\sum_{l'} \exp[score(l' \mid s)]}$$

where l' represents the set of all possible label sequences. Thus, we can find the desired label sequence which gives the highest conditional probability.

With respect to our model, there exists a hidden layer between the BLSTM layer and the CRF layer. The hidden layer integrates the features that we get from forward direction $\overrightarrow{h_i}$ and the backward direction $\overleftarrow{h_i}$ of the BLSTM.

$$h_i = tanh(W[\overrightarrow{h_i} \oplus \overleftarrow{h_i}] + b)$$

The CRF takes the integrated feature representation from the hidden layer as input. Then, we compute the score of each sentence and the conditional probability of the label sequence given the sentence. The model is trained by maximizing the following likelihood training function:

$$S(\Theta) = \frac{1}{M} \sum_{i=1}^{M} \log P(l_i \mid s_i) + \frac{\lambda}{2} ||\Theta||^2$$

where $P(l_i \mid s_i)$ denotes the probability of the label sequence l_i given the sentence s_i from the training set. λ is the regularization hyperparameter used for preventing overfitting and Θ denotes the set of all model parameters.

One of the problems we are facing is that since we are working with Twitter data-set, there is not enough labeled data that we can utilize for the supervised-training with RNN. Therefore, we implement a semi-supervised training algorithm. The idea is to train the network with a few labeled data and at the same time use the network to perform inference to generate some labels for the unlabeled data. We union the labeled data-set with the newly generated labels

Algorithm 1: Semi-Supervised Learning (Zhu et al. 2020)

input : Labeled Data Set L, Unlabeled Data Set U, Probability Threshold p, Model Parameters Update Threshold τ, Maximum of Training Loops MAX_ITER
output: Model Parameters Θ
Initialize Model Parameters Θ ;
Define Confidence Set $C = \emptyset$;
for $i = 1...MAX\_ITER$ **do**
  Train $model_i$ using labeled data set L ;
  Break if the update of Θ is less than τ ;
  Predict the unlabeled data set U using $model_i$;
  for *each unlabeled data* u *in* U **do**
    if $P(y \mid u) > p$ **then**
      $C \leftarrow C \cup u$;
    end
  end
  $U \leftarrow U - C$;
  $L \leftarrow L \cup C_l$; C_l is the labelled set using $model_i$;
  $C \leftarrow \emptyset$;
end

whose confidence level passes a given threshold. This enlarges the amount of labeled data without much human supervision, and we iteratively do this to utilize the unlabeled data. Algorithm 1 gives the pseudo-code of this procedure.

As the baseline measure, we adapt the TF-IDF method, which is a commonly used statistical measure in the field of text mining and information retrieval. It measures the importance of a word to a document in a large collection of documents. The general idea is that if a word appears many times in one document, i.e. the term frequency is high, but it does not appear frequently in the entire collection of documents, i.e. the document frequency is low, then the word is potentially a keyword for the document (Jones 2004). Hence the name Term Frequency-Inverse Document Frequency (TF-IDF).

The result of TF-IDF is the product of Normalized Term Frequency and Inverses Document Frequency. Normalized Term Frequency is a measure of the frequency of the word in one document. We count the number of occurrence of the word in the document, and then divide the number by the total number of words in the document. By such division, we take the length of documents into consideration, since words tend to appear more often in longer texts. On the other hand, Inverses Document Frequency measures the importance of the word over the entire document collection (Kaur and Gupta 2010). For example, terms like "the", "is", "of" that appears plenty of times in all documents should not be considered as a keyword. Therefore, we want to give lower weights to these frequent words, and higher weights to the rare words. The Inverses Document Frequency is calculated as the ratio of the total number of documents in the collection over the number documents that contains the word, then taking the logarithm of this ratio. (Kaur and Gupta 2010)

In symbolic representation, TF-IDF could be computed as follows:

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i})$$

where $tf_{i,j}$ demotes the number of occurrences of word i in document j, df_i represents the number of documents containing word i and N is the total number of documents. (Kaur and Gupta 2010)

We choose to use TF-IDF as the baseline measure because it is commonly used in scientific research under the subject of keyword extraction from texts, and lots of the results have been made available. Additionally, it is easy to implement and compute, so we may also compute the TF-IDF value of our data-set within a reasonable period of time.

Regarding the data-set, we first adapt the data-set used in "*Automatic Keyword Extraction on Twitter*" (Marujo et al. 2015). It is a labeled data set with 1827 tweets, in which each tweet is labelled with a sequence of keywords for that tweet, and the sequence is ordered based on the number of votes by a group of linguistic experts on each keyword. There are 2-5 keywords per tweet in average, depending on the length of the tweet. Hence, there are three variables in this data-set, which are the text of the tweet, the keywords that this tweet contains, and the number of votes for each keyword. We choose this data-set primarily because it was well-labeled and unbiased: the tweets in the data-set covers a vast variety of topics, contexts and users.

Additionally, we will also utilize the Twitter API to collect $9,000$ tweets on our own to ensure that the data-set is large enough. We choose to collect the data-set instead of using an existing one so that we have control over the time of the collection, the topics of the tweets, etc. and we would have up-to-date data as well.

The raw data collected from the Twitter API contains a "text" field, which is purely text message, without extra information such as user id, the time of the post, comments, the number of likes and retweets, attachments, etc. With regard to data pre-processing, since we are only analyzing text in English, we filter out all twitter messages that contain languages other than English. For the content of the "text" field, we choose to keep the mentions and tags since they may potentially contain important information and key phrases. On the other hand, we remove any emoji in the text since emoji are represented as unicode which is difficult to be processed, and emoji generally do not provide much information for keyword extraction.

After extracting the data following the guidelines above, we annotate the text by adding POS and dependency taggings using existing library. The processed data is stored in csv files. Each csv file contains one twitter message. Each row in the csv file corresponds to a single word in the message. There are three columns in the csv file, including the word, the POS tagging and the dependency tagging related to this word.

Results

As is explained in the methodologies, we have obtained a relatively small labeled data-set from the work "*Automatic Keyword Extraction on Twitter*" (Marujo et al. 2015) and a

large unlabeled data-set from Twitter API. For the labeled data-set with 1827 entries, we first shuffle the data-set and reserve 827 of them for testing, and use the rest to perform the training. We use a 10-fold cross validation to compare the performance to tune the hyperparameters of the algorithm. Table 1 summarizes the size of the data-set.

	Train	Validation	Test
Labeled Data-set	900	100	827
Unlabeled Data-set	$9,000$	0	0

Table 1: Size of the Data-set

We implement both our algorithm and the TF-IDF approach. We use the labeled testing data-set to evaluate the performance of these two approach.

We first train our algorithm with a semi-supervised manner explained in the methodology part, and use the trained model to perform inference tasks and predict the keyphrases of the test inputs.

Table 2 summarizes the hyperparameters chosen for the training in the original paper (Zhu et al. 2020).

Hyperparameter	Value
Initial Learning Rate	0.01
Dimension of Word Embedding	300
Dimension of POS Embedding	25
Dimension of Dependency Embedding	25
Hidden Layer Size	100

Table 2: Hyperparameters for the training in Zhu's work

These hyperparameters are chosen for a different data-set, and may not generalize well to our usage. Therefore, we adopt these settings as our initial configuration, and we use a 10-fold cross validation to tune the hyperparameters.

AdaGrad is used in the original work (Zhu et al. 2020). In our implementation, we use Adam as the optimizer, since AdaGrad has the limitation of vanishing gradient problem, whereas Adam can generally avoid this issue and give a better performance.

As a performance measurement, we use F-1 score, a common statistical tool to evaluate the performance of a binary classification task. Since all the keywords or keyphrases are extracted from the original text, we can label each word of the text a boolean representing whether this word is a keyword of the text. Then the problem is reduced to a binary classification task.

We use the traditional F-1 score to measure the performance, which is given by the harmonic mean of precision and recall:

$$F = \frac{2}{recall^{-1} + precision^{-1}}$$

where recall and precision are defined by:

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

Another commonly used performance metrics for evaluating classification tasks is accuracy, which is given by

$$accuracy = \frac{true\ positive + true\ negative}{number\ of\ testing\ examples}$$

Using accuracy might seem intuitive, but for our case, the number of keywords is much less than the total number of words in the text, and thus the number of words falling into the two categories in the binary classification problem is unbalanced. F-1 score takes this into account by taking the harmonic mean of the precision and recall. Thus, we choose F-1 score as our performance measurement.

For the base line, we perform TF-IDF on the same testing data-set with respect to the entire training and testing data as the entire collection of texts to compute the Inverse Document Frequency. This is to mimic the effect of the training data-set on TF-IDF, a non-machine-learning algorithm that does not require a training data-set. We also use F-1 score to evaluate the performance of TF-IDF. We will compare the F-1 score of the two approaches.

We expect our method to perform better than TF-IDF. Although TF-IDF works well when extracting keywords from news, articals, scientic papers, etc, we do not expect TF-IDF to produce a reliable result. Since for Twitter data-set, most tweets do not have their keywords repeated since the tweets are generally short. Therefore, comparing the frequency of words based on statistics might not give a desirable result. On the other hand, the neural network is expected to adapt to this and produce better results.

Implementation and Performance (D3)[1]

As is mentioned previously, we obtain a small labeled data-set with 1827 tweets from the work "*Automatic Keyword Extraction on Twitter*" (Marujo et al. 2015) and a large unlabeled data-set with $9,000$ tweets from the Twitter API. We perform a two-stage data pre-processing on the data-set.

Since our data-set is retrieved from various resources, they are in different formats. At stage I, in which we perform raw-data processing, we extract the text and labeled keywords (if available) from the data-set and summarize them into a csv file, in which the first column is the text, and the second column is the labeled keywords of the tweets (if available). For each tweet, keywords are ordered by the number of votes received from a group of linguistic experts.

Due to the nature of tweets which are generally informal and casual, slangs, abbreviations, hashtags, emojis, typos, and smiley texts, etc. can frequently appear in the tweets. We perform data cleaning with respect to these informal expressions in the tweets by utilizing the `tweet-preprocessor`, which is an open-source python library[2]. At this stage, we tokenize the tweets into lists of

[1]The code is available at `https://github.com/g95wang/twitter_keyword_extraction`

[2]This library is available at `https://pypi.org/project/tweet-preprocessor/`

words, and perform data cleaning on the words. In particular, the symbol of compound (#) in hash tags, emojis, smiley texts (such as :)) and urls are all removed. In addition, we also label each word by their POS and dependency taggings by utilizing an open-source library called `spacy`[3]. In case Twitter users make typos on tweets, each word is converted to the most similar word in a standard English dictionary, and the similarity between two words is measured based on the overlapping of their characters.

Throughout data exploration, we find that the labeled data-set from the work "*Automatic Keyword Extraction on Twitter*" (Marujo et al. 2015) provides a relatively large list of keywords for each tweet. A word is deemed as a keyword if it is voted by at least three linguistic experts. In the case where the group of linguistic experts has a wide range of opinions on keywords, the list of keywords for the tweet get relatively large, and some of the keywords in the list are not representative enough. Hence, during stage II of preprocessing of the data-set, we trim the keyword list to keep it of appropriate length. In particular, we make sure that the number of keywords for a tweet is no more than 10% of the number of valid tokens obtained after the cleaning on the raw tweets by keeping the keywords with the votes from the majority of the group of linguistic experts.

In the implementation stage, we first implement TF-IDF as our baseline method. The implementation of TF-IDF is broken down into two phases. In the first phase, we iterate through all tweets in both the labeled and unlabeled data-set and build a mapping which keeps track of the number of documents with the term t in it, where a term represents a word in the tweets. Then in the second phase, we process each tweet for the second time. We count the number of occurrences of the term t in the current tweet and calculate the TF-IDF score using the mapping we build in the first phase, with the formula provided in the previous section. If the TF-IDF score of this term t is higher than a given threshold, which is a hyperparameter of the algorithm, then we classify it as a keyword.

Using the labeled data-set, we evaluate the performance of TF-IDF by counting the number of true positives, false positives and false negatives. We use these numbers to compute the precision, recall and the F-1 score.

Regarding the TF-IDF threshold, we test the performance of TF-IDF with a list of possible threshold values, and choose the threshold with the highest F-1 score. Figure 3 shows the change in value of the precision, recall, and F-1 score as we change the TF-IDF threshold. The final threshold we choose is 0.295.

For the RNN approach, we adopt the model designed in the paper (Zhu et al. 2020) which is illustrated in Figure 1 with slight modifications on the last layer. Since we want to extract keywords instead of keyphrases from the tweets, CRF computation is not necessary, and a binary classification task is sufficient. We use the sigmoid function as the activation function for the last layer of output, and use that to label whether the word in the sequence is a keyword or not. We label each keyword of the sequence as 1, and 0 for all other words. We use the binary cross entropy to compute the loss of the prediction with respect to the given label sequence.

[3]This library is available at `https://spacy.io/models/en`

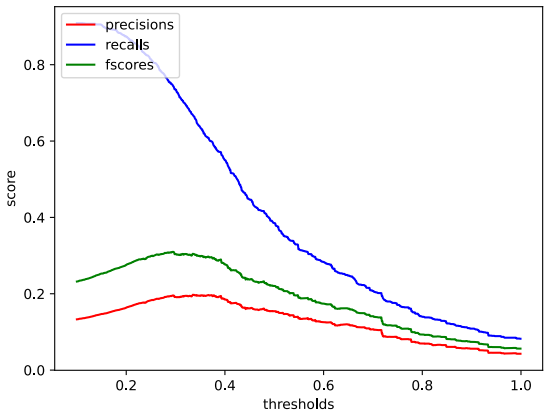

Figure 2: TF-IDF Score with respect to different settings of thresholds. TF-IDF achieves optimal performance at the threshold 0.295

We train the model with two different approaches. First, we perform a traditional supervised learning using the labeled data-set, with the data-set partitioning shown in Table 1. As is explained in previous sections, this data-set is small and might not be sufficient for the training, so we want to exploit the unlabeled data-set. Therefore, as the second approach, we implement the semi-supervised algorithm described in Algorithm 1. As is shown previously, during each epoch of the training, after training on the training data-set, we append unlabeled tweets with a confidence score of the prediction passing a given threshold to the training set. This action enlarges the training set in each epoch, and unlabeled data is utilized in this algorithm.

Figure 3 shows the training and validation curves for the precision, recall and F-1 score with respect to the number of epochs that has been completed in the training process. The validation F-1 score increases at first, where the model is getting more accurate, and then decreases as expected, where the model is overfitting the data. We take the model where the validation F-1 score is the highest, and use this model to perform inference tasks on the test data-set. The result of the test performance is summarized in Table 4.

Notice that the F-1 score of the supervised approach is very close to the one with TF-IDF. This is mostly due to the fact that we are only using 900 tweets to train the model, which is generally a rather small number for the model to be trained. However, the precision for the machine-learning approach is much higher than TF-IDF, and the recall is much lower. For Twitter data-set, most tweets are short, and many keywords only appear once in the tweet. Since TF-IDF extracts the keywords by the relative frequency of each word, it cannot properly recognize this kind of keywords. Therefore, when we select the best threshold, the algorithm would

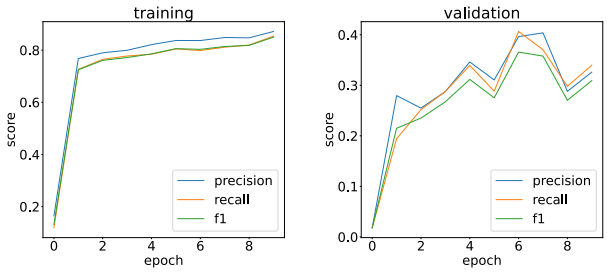

Figure 3: Performance of Supervised RNN Model. Each figure shows the performance in terms of precision, recall, and F-1 score with respect to the number of epochs that has been completed in the training process. Left shows the performance corresponding to the training set, and right shows the performance corresponding to the validation set

try to give an "optimistic" model where most of the words are labeled as keywords. In this way, the recall is high, resulting in a higher F-1 score. This means that although the F-1 score for TF-IDF is high, this approach is actually not very informative, since it would label much more words as keywords than it should. On the contrary, the precision and recall for RNN are more consistent, meaning that the number of keywords generated by RNN is consistent with the labeled data-set.

Semi-supervised learning performs better than supervised learning approach as expected, since it can utilize the much larger unlabeled data-set. This result shows that when we do not have a large labeled data-set, we can still use it to perform the training along with some unlabeled data-set to further improve the model. One of the problems with semi-supervised learning is that we need a threshold of the confidence for the unlabeled data to be added to our training set. After each training epoch, we perform inference tasks on all the unlabeled data-set, and based on the prediction, we compute the confidence score describing how confident we are regarding the prediction. We use the negative log likelihood function to compute this score, and we choose the threshold hyperparameter by running the training multiple times and compare the test performance of each settings of the hyperparameter to choose the best one. Table 3 summarizes the results.

Threshold	Precision	Recall	F-1 Score
0.020	0.352	0.326	0.339
0.010	**0.368**	**0.349**	**0.358**
0.005	0.318	0.346	0.331
0.001	0.309	0.310	0.309

Table 3: Test result of semi-supervised learning with different thresholds. With threshold being 0.005, the model achieves the highest precision, recall and F-1 score.

When the threshold is chosen as 0.005, the model gives the best performance. We use this threshold for our semi-supervised learning algorithm. The training and validation precision, recall and F-1 curves are provided in Figure 4. We notice that compared to supervised-learning curves, training accuracy begins to drop after some epoch. This is due to the fact that more data appended to the training data-set as the model matures. When we train the new data, the training accuracy is rather low, resulting in a higher loss.

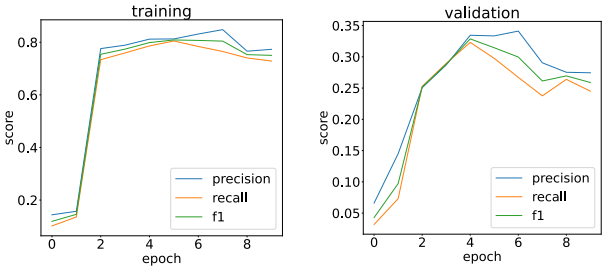

Figure 4: Performance of Semi-supervised RNN Model. Each figure shows the performance in terms of precision, recall, and F-1 score with respect to the number of epochs that has been completed in the training process. Left shows the performance corresponding to the training set, and right shows the performance corresponding to the validation set

The results of is summarized in Table 4.

Techniques	Precision	Recall	F-1 Score
TF-IDF	0.196	**0.744**	0.310
Supervised RNN	0.310	0.311	0.310
Semi-supervised RNN	**0.368**	0.349	**0.358**

Table 4: Test results of identifying keywords of tweets using TD-IDF, Supervised RNN Model, and Semi-supervised RNN Model. Precision, recall, and F-1 score for each techniques of identifying keywords in tweets are presented. Utilizing TF-IDF and supervised RNN model shows similar results in F-1 score. However, we achieve 0.358 on F-1 score by using semi-supervised RNN model, and its results of precision and recall are generally consistent.

We conclude that the semi-supervised method gives the best performance amongst the three methods shown above.

Lessons Learned (D3)

In this project, we get to explore the field of Natural Language Processing. The research gives us the opportunity to witness how the problem of extracting keywords is tackled by other researchers, and how each solution is developed and evolved over time. We become familiar with many aspects of NLP including data retrieval, text cleaning, text preprocessing, POS tagging, dependency tagging, and Machine Learning models such as RNN, LSTM, MLP, SVM etc. that are commonly used for NLP.

During implementation, we encounter and try to solve many problems which are specifically caused by twitter messages. For instance, it is hard to find a large twitter data set with labeled keywords. Therefore, we adjust our implementation plan from a supervised learning to a semi-supervised learning. Another challenge is that slangs, abbreviations, and typos appear frequently in tweets and these

"user-created" words cannot be encoded to vectors accurately and some will introduce errors when analyzing the texts. Thus, we find a standard English dictionary and convert all "user-created" words to the most similar word that can be found in the dictionary. Moreover, we also get many hands-on experience of utilizing existing libraries that are commonly used for NLP and Machine Learning when processing data and building the model.

At evaluation phase, we learn different types of evaluation metrics and choose F-1 score to evaluate the performance instead of simply measuring the accuracy of the model prediction. We get to analyze and compare the performance of the model in detail. For instance, we cannot simply evaluate the model by comparing the F-1 score. The value of the recall and precision also reflect the model behaviour. The process of reasoning about the performance and adjusting the model to achieve better result is also valuable. It is important to consider not only the model itself, but also other factors including the feature of twitter messages, the labeled data-set, the pre-processing of the input data, etc.

Discussion

Conclusion

References

[Benayed et al. 2003] Benayed, Y.; Fohr, D.; Haton, J. P.; and Chollet, G. 2003. Confidence measures for keyword spotting using support vector machines. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, I–I. IEEE.

[Chomsky 1986] Chomsky, N. 1986. *Knowledge of Language. Its Nature, Origin, and Use*. Convergence. New York/Westport/London: Praeger.

[Colby 1974] Colby, K. M. 1974. Ten criticisms of parry. *SIGART Bull.* (48):5–9.

[Jones 1972] Jones, K. S. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

[Jones 2004] Jones, K. 2004. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation* 60:493–502.

[Kaur and Gupta 2010] Kaur, J., and Gupta, V. 2010. Effective approaches for extraction of keywords. *International Journal of Computer Science Issues (IJCSI)* 7(6):144.

[Liu et al. 2009] Liu, Z.; Li, P.; Zheng, Y.; and Sun, M. 2009. Clustering to find exemplar terms for keyphrase extraction. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, 257–266.

[Marujo et al. 2015] Marujo, L.; Ling, W.; Trancoso, I.; Dyer, C.; Black, A. W.; Gershman, A.; de Matos, D. M.; Neto, J. P.; and Carbonell, J. G. 2015. Automatic keyword extraction on twitter. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 637–643.

[Orphanos et al. 1999] Orphanos, G.; Kalles, D.; Papagelis, T.; and Christodoulakis, D. 1999. Decision trees and nlp: A case study in pos tagging.

[Tenor 2018] Tenor. 2018. Tenor hits 12 billion monthly gif searches as it ramps global expansion with first international advertisers content partners.

[TURING 1950] TURING, A. M. 1950. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind* LIX(236):433–460.

[Weizenbaum 1966] Weizenbaum, J. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9(1):36–45.

[Winograd 1971] Winograd, T. 1971. *Procedures as a representation of data in a computer program for understanding natural language*. PhD thesis, Massachusets Institute of Technology.

[Wu et al. 2005] Wu, Y.-f. B.; Li, Q.; Bot, R. S.; and Chen, X. 2005. Domain-specific keyphrase extraction. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, 283–284.

[Zhang et al. 2016] Zhang, Q.; Wang, Y.; Gong, Y.; and Huang, X.-J. 2016. Keyphrase extraction using deep recurrent neural networks on twitter. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, 836–845.

[Zhu et al. 2020] Zhu, X.; Lyu, C.; Ji, D.; Liao, H.; and Li, F. 2020. Deep neural model with self-training for scientific keyphrase extraction. *Plos one* 15(5):e0232547.