Can Emoticons be Used to Predict Sentiment

University of Evansville
Keenen Cates - kc235@evansville.edu
Dr. Pengcheng Xiao - px3@evansville.edu

March 2017

Abstract

Getting a machine to understand the meaning of language is a largely important goal to a wide variety of fields, from advertising to entertainment. We focus on Youtube comments from the top two-hundred trending videos as a source of user text data. Previous Sentiment Analysis Models focus on using hand-labelled data or predetermined lexicons. Our goal is to train a model to label comment sentiment with emoticons by training on other user-generated comments containing emoticons.

1 Introduction

Sentiment analysis is a branch of natural language processing that involves trying to understand the underlying sentiment and emotion behind language. For example, "Have a great day" has a positive sentiment, and "Have a bad day" has a negative sentiment. Current state of the art techniques for modelling sentiment in language involve using machine learning and deep neural networks to classify the sentiment of language. For example, SemEval is a yearly contest for trying to classify tweets as Positive, Negative, or Neutral. Its findings advance the field of sentiment analysis and machine learning [2].

1.1 Objectives

Our focus is on another major social platform, Youtube, which garners hundreds of thousands of comments and other user generated statistics. User data yields important results in the fields of social sciences. In particular we are interested in the top trending youtube videos, and aim to identify sentiment of commenters by suggesting what emoticon a user might use with their comments. We suggest emoticons give insight into the sentiment of the user, and the emoticon's pictographic nature gives us a better language to indicate emotion. Using the subset of comments with emoticons we engineered a labelled dataset of comments and emoticons. Our models take advantage of this labelling to model the emoticon lexicon. This is further used to suggest what emoticons might accompany a comment[1]. Using this dataset and the models we have create, we hope to answer whether or not we can accurately predict what emoticon a user might use.

1.2 Literature Review

Sentiment Analysis drives many industries and being able to correctly identify sentiment in a Youtube comment would allow automated systems to moderate comments or correctly recommend media or advertisements to users. In general, there are two methods that Natural Language Processing researchers use for Sentiment Analysis; Lexicon based and Machine Learning based. Sentiment Analysis is a fairly robust field, and has consistently seen interest since its conception. This field has increased exponentially with the surge in data seen with the rise of the internet, in many cases the amount of data is intractable. Social platforms such as Youtube, by themselves generate more data than any one human could analyze. Therefore a system of Natural Language Processing (NLP) is required to deal with the sheer volume of data.

Natural Language Processing can be considered a subset of cognitive science or computer science. The concept of natural language processing originally came about in the mid-20th century. The initial motivation was language translation [3]. Natural Language Processing naturally lends itself to the field of Artificial Intelligence, as there is a strong desire for agents that can understand human language; for example, a chat bot. Sentiment Analysis did not pull much attention until the early 2000s. The natural language processing systems that were developed at first were only applicable to narrow subject areas, such as answering questions with information from a database about moon rocks, or answering questions from a manual on airplane maintenance [14]. The explosion of social data quickly created a necessity to autonomously understand language sentiment. Especially with the ubiquitous nature of social media in recent years, the field of sentiment analysis has become more and more applicable to many fields. It has been one of the most active areas of research in the field of natural language processing since the turn of the century [18].

There are many commercial applications. It may have significant effects for the fields of management, political science, economics, and other social sciences, among others. [14]. Sentiment analysis, also known as opinion mining, refers to the process of creating automatic tools or systems which can derive subjective information from text in natural (human) languages, as opposed to computer codes. The subjective information most commonly desired by researchers are opinions and sentiments, hence the name sentiment analysis. Sentiment analysis, while originally only practiced by computer scientists, has become widely used by the management scientists and the social sciences. Microsoft, Google, Hewlett-Packard, IBM, and others have created their own systems for sentiment analysis. Before the turn of the century, there were previous developments in what would later become the field of sentiment analysis. (Source 8) provided a way to model the affective tone of an entire document based on the "semantic differential scores" of each of the words in the document. The semantic meanings and scores were derived from a 1965 study by Heise.According to Lee and Pang, the year 2001 or so marked an explosion of research in sentiment analysis. This increase in the study of this topic was partially attributed to the increasing popularity of machine learning models, and the availability of training sets with which machine learning models could be trained [19]. Peter D. Turney (2002) used an algorithm based on parts-of-speech tagging and semantic orientation in order to classify online reviews as recommended or not recommended [20]. Anderson and McMaster (1982) used machine learning techniques such as Support Vector Machines and Naïve Bayes in order to classify the sentiment of movie reviews [9]. In 2003, Dave, Lawrence, and Pennock classified polarity of web reviews based on several n-gram methods. It was not as accurate when applied to individual sentences because it was developed with the purpose of classifying reviews which normally contained multiple sentences [12]. In 2004, Hu and Liu used a method that could predict the sentimental orientation of opinion words and therefore the opinion orientation of a sentence. It was an unsupervised method and did not require a corpus, and was loosely based off the work of Dave, Lawrence and Pennock. It returned the sentiments at the sentence level instead of at the entire review at once. Then it combined the sentence-level sentiments to give a summary of the entire review [13]. Moraes, Valiati, and Neto (2013) showed the effectiveness of machine learning processes as opposed to lexicon-based models. They empirically compared the Support Vector Machines and Artificial Neural Network machine learning methods for sentiment analysis and found that the Artificial Neural Networks performed better [17]. In 2015, Wang et al. showed the effectiveness of Long short-term memory recurrent neural networks for sentiment analysis by predicting the sentiments of tweets [21].

1.3 Sentiment Lexicon

The lexicon method splits input text into many individual words or phrases called tokens. Then, it creates a table of these tokens and records the number of times each token shows up in the text. The resulting tally is called a "Bag of Words" model. Once this process is done, another tool called "Sentiment Lexicon" is used for computing the classification of the bag of tokens we mentioned above. The Sentiment Lexicon has the sentiment values, which can be just positive or negative numbers or some other value-representations, like vectors, that are pre-recorded for each token. This can be done either manually or by some machine learning techniques. Once we have the input text tokenized and a suitable Sentiment Lexicon, the final task is to design a function to compute the final sentiment. The simplest way to compute the final sentiment is to sum each token's sentiment values together. The lexicon method is a traditional way to deal with natural language processing problems, and it has a good theoretical basis. Many people are still using and studying this method in spite of its origins in the 1960s. However, it does have some drawbacks such as ignoring the importance of integrality and continuity of the text. We know that the meaning of a sentence highly depends on the order of words and context; these should not be ignored if we want a real intelligent sentiment processing system [5].

1.4 Machine Learning

In the Machine Learning technique of sentiment analysis the classification algorithm uses a training set to learn a model based on features in the set. This makes a more nuanced classification possible and can help with ambiguous words or interpretations that vary by context. A method of feature extraction must be chosen. Some of these methods include n-grams, which are sets of words that contain n words each. Others use parts-of-speech information, emotional, affective, or semantic data. One of the disadvantages of the machine learning method is that it requires a large set of labelled data to be used as the training set. It is simpler to use the lexicon-based method unless a suitable training set is available. [3]

We will need to classify the sentiments of the emoticons manually in order to prepare them for use in our analysis. Once that is done, we can compile our training set using the comments in the data that already contain emoticons, using the sentiments of each emoticon. Then our model will be able to classify and assign an emoticon to each comment in the data set that does not already contain one. Recurrent Neural Networks have had a great deal of success in the Natural Language Processing Realm. The reason is that text data is highly sequential, for example, the word "day" doesn't mean much unless you know the words that came before it; i.e "Have a great day." RNNs have pushed the state of the art of previous architectures in short-length text data [7].

Given previous attempts to model sentiment have not thoroughly explored emoticons, we hope to answer the question of whether or not we can accurately recommend emoticons that might accompany a piece of text. Once we have answered this, further research can make attempts to analyze sentiment with emoticons on a machine.

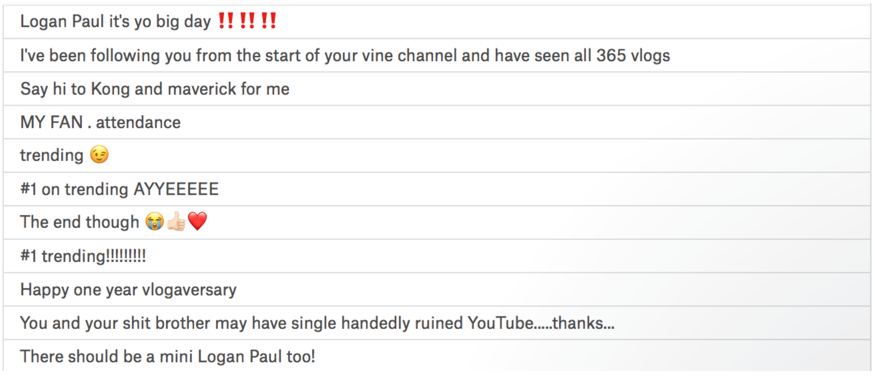
2 Methodology

2.1 Data

To get our data, we used the Data Science Competition Website Kaggle. On this website, people share datasets, competitions, and tutorials. We found a dataset containing comments from the top 200 trending Youtube videos. The author of this dataset obtained the data through Youtube's publicly available API, which allows developers to easily query for data on Youtube. The data itself contains profanity, non-sensical text, and in general is noisy. The data itself could be generated by bots, and we do no vetting to determine whether a comment actually comes form a human. The noisiness of the data might prevent us from training a successful model; however, we assume that the large amount of data will help our models perform well in spite of the low quality of data.

In order to answer the question of whether or not a model could recommend emoticons, we created 2 models that attempt to perform this recommendation. We also created a simple dummy model for purposes of comparison. We have roughly three-hundred thousand comments with emoticons, and use that to boostrap a dataset of comments with labels. More data is desirable, but this is a fairly large corpus for initial research.

In total, there are 691, 388 rows in the dataset. A large proportion of them contain emoticons, (more than 200, 000), so there is a quite a bit of data, and it would be fairly straightforward to access the Youtube API and get more if needed. This means I have as much data as I could possibly want, and more if needed. As for features, I will only use the text, likes, reply threads, and so on will be ignored in this phase of the project. On average, each text is 15 words long. The figure belows shows some examples of how the data looks.



Logan Paul it's yo big day ‼️‼️‼️
I've been following you from the start of your vine channel and have seen all 365 vlogs
Say hi to Kong and maverick for me
MY FAN . attendance
trending 😉
#1 on trending AYYEEEEE
The end though 😭👍🏻❤️
#1 trending!!!!!!!!!
Happy one year vlogaversary
You and your shit brother may have single handedly ruined YouTube.....thanks...
There should be a mini Logan Paul too!

Table 1: Example unprocessed data

2.2 Evaluation Metrics

The models will be evaluated using a holdout set of data, in which each will recommend five emoticons that might accompany a text. If at least one recommendation is an emoticons that occurs in the validation comments, then

I will consider it to be a "correct" guess. Accuracy is then the number of correct guesses divided by total guesses. Keras calls this accuracy "top k categorical accuracy", and will be implemented for our models. Mathematically, this would look something like this where matching $x \in Comments$ and $y \in Labels$ and $score(x) = 1$ if any $p \in argmax_{k=5}(predict\_labels(x))$ is in y, else $score(x) = 0$. $predict\_labels(x)$ would return the probabilities of each output class occurring. Then the accuracy of the model would be $\frac{\sum_{i=0}^{N}(score(x_i))}{N}$ where $x_i \in Comments$ and $N = \mid Comments \mid$.

One consideration is that the distribution of emoticons occurring in the corpus of data is highly skewed; this would be good reason to suggest F1 scores and might be better for future analysis. However, we chose this evaluation metric because it more closely resembles the question we are asking. MatPlotLib has trouble rendering many of the unicode emoticons; however, some the emoticons do render. The important thing to note is that the distribution is indeed skewed.

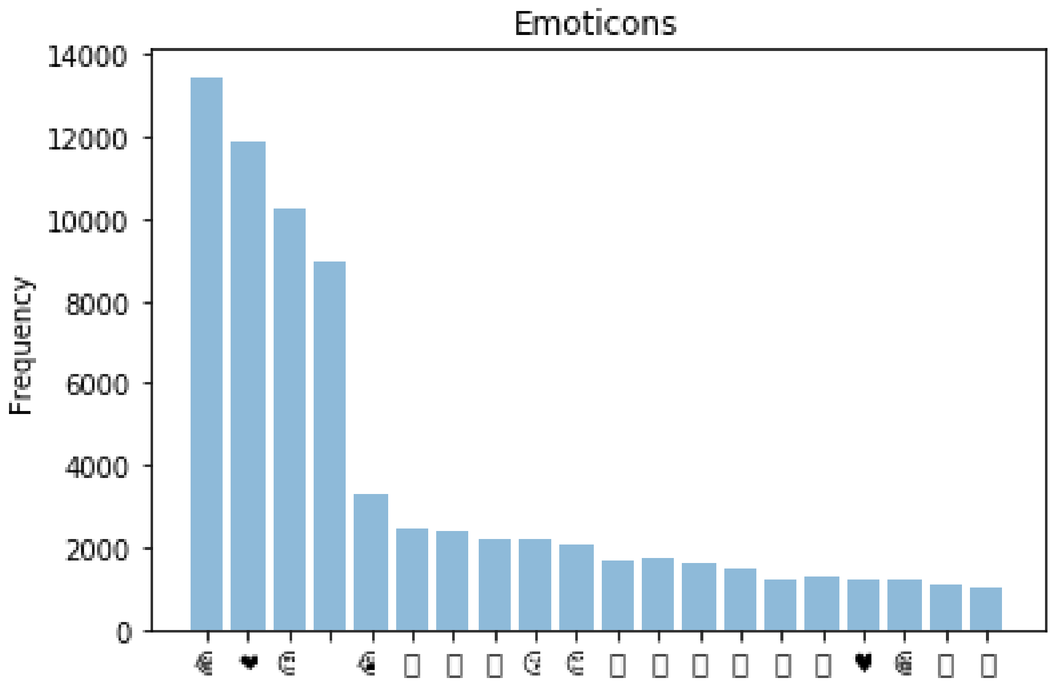


Figure 1: Distribution of a subset of Emoticons

2.3 Analysis Plan

In order to compare the performance of our model, we created a holdout set of data meant for only validation of accuracy. We also defined what a prediction would be for each model, each model would output its top five highest predictions. If any of those predictions are in the output validation set, then we considered it an accurate prediction. Then in order to analyze the dataset, we will compute the prediction accuracy of each model and compare those scores. One might also consider looking at the training accuracy of each model; however, these scores are not directly comparable, so we ignore them except for the purposes of optimizing the model.

2.4 Approach

In our approach, we had to make a few crucial assumptions and simplifications to contextualize our problem. Firstly, our dataset involved input data that would lead to a multi-class, multi-label classification problem. For example, a users can add hundreds of the same emoticon or many different emoticons. As a preprocessing step, we narrowed down these classes to the unique emoticons that show up in a comment, and in the RNN model unrolled the data set to have a single label; effectively making the problem a multi-class, single-label classification problem. Figure 1, displays how each comment gets unrolled into multiple data points with single labels. For the Bayesian Model, we can still deal multi-class, multi-label classification fairly straightforwardly.

I loved this video!	x x y
I loved this video!	x
I loved this video!	y

Figure 2: Unrolling of data labels

The other assumption exists only for our Naive Bayes Model, and it is that all words in the comments are independent. This assumption is difficult to back up, and it is not clear whether there is mutual dependence or mutual exclusivity between words. However, our Recurrent Neural Network does not have this limitation because it can model the entire sequence.

2.5 Preprocessing

One of the most important steps is the preprocessing stage. This is done before all models are trained. We first separate the data into comments with emoticons and comments without emoticons. We then make all comments lowercase and afterwards normalize our comments on both by creating a dictionary of punctuation to tokens, and a dictionary of word counts over all comments that use the ordering of each word as its embedding. Figure 3 shows an example of how the dictionaries are used to tokenize a comment. Word embedding is a way to turn English words into numbers, and this often needs to be done to for models that require quantitative data.

Punctuation to Token Dictionary

!	<exclaim>

Word to Integer Dictionary

i	0
love	1
this	2
class	3
<exclaim>	4

Tokenization Process

1).	"I love this class!"
2).	"i", "love", "this", "class", "¡exclaim¿"
3).	0, 1, 2, 3, 4

Figure 3: Tokenization Process

A similar process is used to encode the emoticons, we use a dictionary to encode them as integers. Preprocessing the comments in this way gives us a normalized integer sequence, which deals with comments that might have different capitalizations of words. Lastly, the data labels are unrolled for the RNN model as displayed by Figure 2.

2.6 Dummy Model

For purposes of comparison, we created a very simple model that always predicts the emoticons with the top five highest probabilities. The motivation behind this, is that it gives us a baseline score to beat, in addition to our Bayesian model. If we can do significantly better than this, then we know that the models have potential. To implement this model, we will simply make a table of counts and take the elements with the five highest counts.

2.7 Naive Bayes Model

Our second model uses Bayesian Statistics that creates tables of posterior probabilities for each class given a word using Bayes rule. Naive Bayes is a conditional probability model, and given some instance to be classified, represented by a vector of features:

$$x = (x_1, ..., x_n)$$

We then compute the probability of each output class using conditional probability.

$$p(C_k|x_1, ..., x_n)$$

Since n, can be large making this model less tractable we need to reformulate our model using Bayes Rule. In plain english,

$$posterior = \frac{prior \cdot likelihood}{evidence}$$

And symbolically,

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

In practice, the numerator is the most import part as the denominator does not depend on effectively making it a constant. The numerator is equivalent to the joint probability model meaning we can replace the numerator with,

$$p(C_k, x_1, ..., x_n)$$

We can then rewrite the numerator using the chain rule for repeated applications of conditional probability, derivation is in appendix 1. Then we add the naive assumption of conditional independence, allowing use to further simplify our model to:

$$p(C_k|x_1, ..., x_n) = \frac{1}{Z}p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

Where Z is:

$$Z = p(x) = \sum_k p(C_k)p(x|C_k)$$

Which is the scaling factor dependent on the instance. The derivation is in appendix 2. In order to make a classifier, we would generally take the argmax of the simplified model without Z , but in our case we take the top five arguments as our program is recommending multiple emoticons that might be appropriate[8]. We implement this model in python and the model follows figure 3.

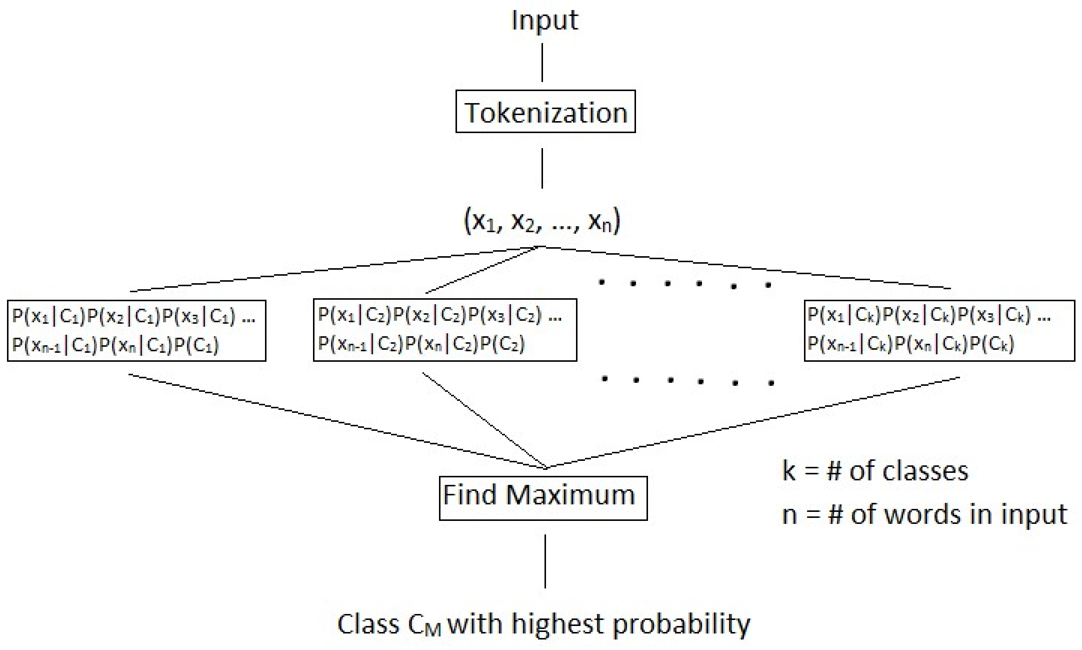

Input
Tokenization
(x1, x2, ..., xn)
P(x1|C1)P(x2|C1)P(x3|C1) ... P(xn-1|C1)P(xn|C1)P(C1)
P(x1|C2)P(x2|C2)P(x3|C2) ... P(xn-1|C2)P(xn|C2)P(C2)
P(x1|Ck)P(x2|Ck)P(x3|Ck) ... P(xn-1|Ck)P(xn|Ck)P(Ck)
Find Maximum
k = # of classes
n = # of words in input
Class CM with highest probability

Figure 4: Naive Bayes Model

Another problem is that we have to deal with words that never show up in our corpus of texts. In order to deal with this, we smooth the probabilities. To do this, we make any word or class that doesn't show up have a very small probability that is close, but not zero. Otherwise, the probability would zero out when words are not in the corpus.

2.8 Recurrent Neural Network

Our third and final model, is a recurrent neural network and our architecture is as follows in Figure 3.

Input
Embedding Layer
LSTM Layer
LSTM Layer
LSTM Layer
Fully-Connected Layer
Output layer

Figure 5: RNN Architecture

Recurrent Neural networks are a class of neural networks that form a directed cycle, allowing them to take time into account, or a notion of memory. This allows for the RNN to be suited to predicted arbitrary sequences by taking advantage of their memories.
The label data also undergoes another transformation before the RNN begins the learning process. Since the emoticons are encoded using an ordinal number, the integer representation doesn't quite make sense as one emoticon is not greater than another. To rectify this, we represent this integer as a one-hot vector, essentially we take a fixed-length vector that is the size of the total number of output classes. Then the integer is used as an index of the "hot" class. Figure 4 gives a small example of encoding a small class space.

Unique Classes: 0, 1, 2, 3, 4
Number of Classes: 5
Example Class: 2

One-Hot Encoding of Example
1). Create zeroed vector:
<0, 0, 0, 0, 0>

2). Make the element at the index equal to the Example Class 'hot':
<0, 0, 1, 0, 0>

Figure 6: One-Hot Encoding

One of the major features of this model is the stacked LSTM layers. This architecture allows us to better model hierarchical elements of language. This means each layer will represent progressively complex parts of the hierarchy. One might imagine this in terms of the composition of the human face. For example, the most basic element is an edge. Then a more complex step would be individual elements of the face such as a nose or mouth. Then the most complex part would be the entire face, and the composition of its requisite parts.
The LSTM itself is able to remember previous contexts in sentences, meaning we could potentially get more performance via our model becoming better at modelling context.
Our RNN had a much longer time to run, and in order to train the model in which we set to run for twenty epochs. The Neural Network was then trained on a GPU using Floyd Hub, a platform for running deep learning projects. The expense was roughly 14 dollars, as a we subscribed to the Data Science plan which gave us 10 hours of GPU time which we used for experimentation on multiple occasions. The price was remarkably cheap compared to other platforms such as Amazon. Usage of FloydHub is remarkably simply, and resembles version control programs such as git. One simply uploads their code to the website using command line tools, and are given an interface to interact with their instance. This service was worthwhile to learn because it abstracted away elements such as infrastructure, version control, and storage and we could focus on the problem.
In addition to our baseline architecture, we also preform dropout on each layer, which helps prevent against training bias because the network probabilistic "drops" some of the weight which forces the network to build redundancies. For the training metric, we implemented the top k categorical accuracy metric listed in the evaluation metrics. For the objective function we found that categorical cross entropy work best which typically works well in multi-class, single-label scenarios.
Using TFLearn, a deep learning library for Python, we implemented the architecture we decided on with relative ease. TFLearn builds on top of TensorFlow, abstracting away many of the more intimate computational components, and allowing the programming to think about the layers and interactions between layers rather than how to build a well known type of layer or cell.

2.9 Implementation

2.9.1 Programming Language Libraries

- **Python 3**
- **TFLearn** a deep learning library featuring a higher-level API for TensorFlow.
- **TensorFlow** a deep learning library

As mentioned throughout the text, the models where implemented using the listed libraries. We did our coding on the website FloydHub via iPython Notebooks, which abstracted away much of the setup. We split our code up into three notebooks, one for preprocessing, Bayesian Model, and RNN. We ran into very few problems implementing our solution; however, some are outlined below.

2.9.2 Problems

- **Bayes Smoothing** We ran into a small hitch with the Bayesian when dealing with querying prior probabilities when certain values did not exist in the data. However, we used a technique to "smooth" the values by assigning a small probability to these values.
- **Skin Tone Modifiers** There are emoticons that exist that modify other emoticons, i.e. allowing one to change the skin tone of the smiley face. We found that these confounded our predictions, and removed them as possible predictions.
- **Finding loss, activation, and metrics** We had to experiment many times to find the best loss, activation, and metric functions for our RNN. This process may be simple trial-and-error as we experienced.

2.10 Refinement

Originally, our RNN model did not preform as well as we had hoped; however, a few optimization to our model vastly impacted our performance. The first model we used was a multi-class, multi-label classifier which performed very poorly. Our RNN had performance at .508 which left much to be desired. We believe the reason for this is that instead of one-hot encoded vector, we had many-hot encoded. This means that the label space would be of order $2^{\#\ of\ emoticons}$. Since this space is extremely large, the model would have trouble representing any reasonable portion of this. For this reason, we needed to unroll data points to preform multi-class, single-label classification. After adjusting our loss function, metric function, and activation function we ended up with much better performance. We believe this to be because of the reduction in potential labels to just # of emoticons. In addition, hyper parameters were adjusted, such as, learning rate and batch size to find out what setting worked best. The best we found was a learning rate of .001 and a batch size of 128.

3 Results

In order to validate the models, we created a holdout set of labelled data that none of the models got to use for training or testing. The accuracy of each model using top k categorical accuracy is in tables 1 and 2.

Model	Accuracy
Dummy	.527
Naive Bayes	.859
RNN	.702

Table 2: Training Accuracy Results

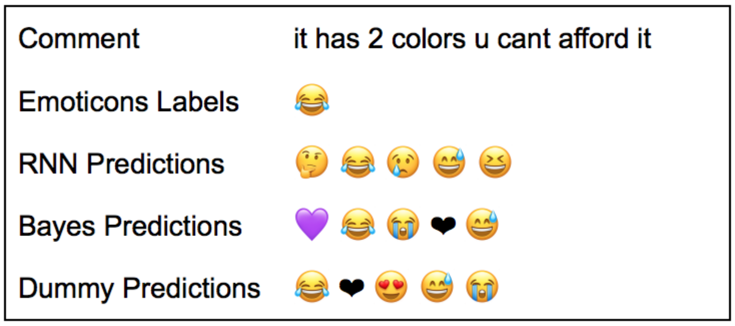
Model	Accuracy
Dummy	.527
Naive Bayes	.548
RNN	.812

Table 3: Validation Accuracy Results

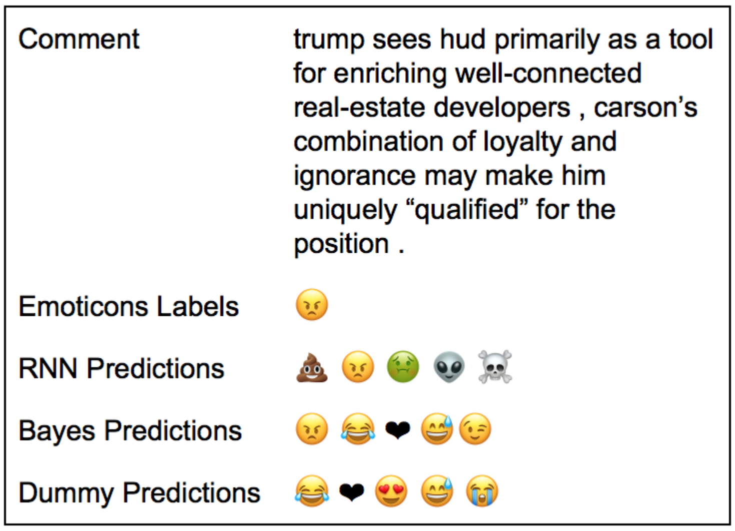
Table 2 gives us a measurement of how well our recommendation engine gives us accurate emoticons to represent our text. Our results do not promote strong confidence in our Naive Bayes Model's ability to recommend emoticons; however, there are some potential improvements to the model such as n-gram modelling. Notably, the Bayesian Model preforms decently on the training data, but generalizes quite poorly and shows signs of over-fitting. The RNN on the other hand, surprisingly preforms slightly worse on training, but preforms much better on the validation set. For whatever reason this phenomenon occurs, it is clear that the model generalizes much better.

3.1 Visualization of Model Functionality

We have a model that could be incorporated into a wide variety of applications; for example, a browser plugin that predicts what emoticons you might put with a comment and assist the user similar to an auto-complete feature. One issue to consider might be the nature of Youtube comments themselves, which might prevent the generalization of this model to other applications. However, the models do show that this sort of functionality is possible. For example, we have pulled some examples from the data and run them through our models to produces the tables below, and the comments themselves seem to be quite different than more formal forms of language.



Comment	it has 2 colors u cant afford it
Emoticons Labels	😂
RNN Predictions	🤔 😂 😢 😅 😆
Bayes Predictions	💜 😂 😭 ❤ 😅
Dummy Predictions	😂 ❤ 😍 😅 😭



Comment	trump sees hud primarily as a tool for enriching well-connected real-estate developers , carson's combination of loyalty and ignorance may make him uniquely "qualified" for the position .
Emoticons Labels	😠
RNN Predictions	💩 😠 🤢 👽 ☠
Bayes Predictions	😠 😂 ❤ 😅 😉
Dummy Predictions	😂 ❤ 😍 😅 😭

Table 4: Example data and predictions

While the machine learning back-end may not be the most sophisticated, the model does a good job in practice of giving recommendations, and we think the model would be good enough to use for applications to be built on top of.

3.2 Limitations

One limitation of our models is that words that do not show up in the Youtube Comment corpus cause issues, as our models have trouble predicting outputs for words that it has never seen. One way to fix this, might be to mine for more Comment data. Some drawbacks of the Naive Bayes Model is that we may not be able to model longer term trends in comments, however with the short length of the comments, this may be a non

issue. We also are limited in our choice of language modelling because we are on the word level. We would likely see improvement by expanding our level of modelling to some type of n-gram. The RNN has limitations on multi-label classification because of the large label space, and this may be hindering its ability to preform optimally. Another limitation might be the actual training time, each epoch costs computational resources which limits our ability to push the model to fulfill its max potential. The model would likely continue to learn and perform better with more training time and data up to a point, meaning ultimately a higher cost for the model when experimenting with this tradeoff. The Bayesian model on the other hand is easy to program with fast run time, and no need to train for hours upon hours.
Another major consideration is that an RNN might be a bad fit. We originally though long term sequential modelling would be important, but it turns out the average comment length is 15 words long. It may be the case that sense the length of texts are so short, that we might have to thoroughly rethink what our strategy would be if this sequential modelling is unimportant.

3.3 Future Work

In order to eliminate the assumption of independence in the Bayesian model, we can add complexity by changing at what level we model the data. To do such we would need to employ a skip-gram or n-gram model that contain larger parts of the sequence data. One might also explore alternative Bayesian Models such as Hidden Markov Models. The same improvements to the data modelling using n-grams would likely improve the quality of the RNN results. The RNN model likely has a great deal of room for improvement, one might experiment with more hyperparameter tuning or modifying the architecture. There are even more powerful models such as CRNNs and GANs that push the state of the art in deep learning. These models would be worth exploring; however, we pushed our newfound deep learning knowledge as far as we could in the time allotted.
Another important consideration is the unrolling of the data. Future work should further explore how to deal with multi-label classification, which would likely involve writing new metric, loss, and activation functions for the neural network model. However, the Naive Bayes Model does not suffer from this limitation.
Future work might also try and further connect the emoticons and sentiment. We hypothesize that emoticons will naturally lend themselves to convert into sentiment classes. However, our current models predict only what emoticon might be used, and the user of the model would have to infer what sentiment the emoticon might convey depending on context.
One might also find more optimizations by adding further preprocessing steps, for example, eliminating common english words that add very little information.

3.4 Reflection

Looking back at the process, here are the steps we took to get to the current models

- **Literature Review** We made sure to have a rough idea of what people in this field have tried, and what the state of the art is.
- **Deciding on a Model** After reviewing the field, we made a decision on what models we wanted to implement which set the tone for preprocessing and implementation.
- **FloydHub** Next we setup our programming environment with cloud computing in mind. It's important to setup an environment such as FloydHub or AWS to minimize training time on a fast gpu. At this step we also made sure to download all the libraries we would need
- **Preprocessing** a large majority of time was spent trying to learn how to deal with the data, and exploring the data itself. We had to go through multiple iterations of embedding and tokenization to find the method that made sense.
- **Model Implementation** After preprocessing our data, this step was fairly straightforward. Most of the time at this step is dealing with edge cases, or optimization of models rather than the actual implementation.
- **Refinement** Refinement may have been the hardest part because we had to make inferences about why our model was not performing up to our desires. It's hard to say what the potential of each model was, so we kept iterating until we had something that seemed substantial.

3.5 Conclusion

Overall, there are many areas for potential improvement, and our work serves as a baseline for recommending emoticons. However, we have begun to answer our original question, it seems plausible the emoticons can be assigned with accuracy to comments as noisy as youtube comments, making it easy for a casual observer to understand the sentiment of a text.

4 Bibliography

1. Hogenboom, Alexander, et al. "Exploiting emoticons in sentiment analysis." Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013.

2. Rosenthal, Sara, Noura Farra, and Preslav Nakov. "SemEval-2017 task 4: Sentiment analysis in Twitter." Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017). 2017.

3. Salas-Zárate, M. P., Medina-Moreira, J., Lagos-Ortiz, K., Luna-Aveiga, H., Rodríguez-García, M. Á., & Valencia-García, R. "Sentiment Analysis on Tweets about Diabetes: An Aspect-Level Approach." Computational & Mathematical Methods In Medicine, 1-9. 2017.

4. Siersdorfer, Stefan, et al. "How useful are your comments?: analyzing and predicting youtube comments and comment ratings." Proceedings of the 19th international conference on World wide web. ACM, 2010.

5. Taboada, Maite, et al. "Lexicon-based methods for sentiment analysis." Computational linguistics 37.2 (2011): 267-307.

6. Kang, Mangi, Jaelim Ahn, and Kichun Lee. "Opinion mining using ensemble text hidden Markov models for text classification." Expert Systems with Applications (2017).

7. Lee, Ji Young, and Franck Dernoncourt. "Sequential short-text classification with recurrent and convolutional neural networks." arXiv preprint arXiv:1603.03827 (2016).

8. "Naive Bayes classifier." Wikipedia, Wikimedia Foundation, 30 Nov. 2017, en.wikipedia.org/wiki/NaiveBayesclassifier.

9. Anderson, C., & McMaster, G. (1982). Computer Assisted Modeling of Affective Tone in Written Documents. Computers and the Humanities, 16(1), 1-9.

10. Brill, E., & Mooney, R. J. (1997). An Overview of Empirical Natural Language Processing. AI Magazine, 18(4), 13.

11. Chipman, S. E. (2017). The Oxford Handbook of Cognitive Science. Oxford: Oxford University Press.

12. Dave, K., Lawrence, S., and Pennock, D. 2003. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. In Proceedings of the 12th International Conference on World Wide Web (WWW '03). ACM, New York, NY, USA, 519-528.

13. Hu, M., & Liu, B. (2004, August). Mining and Summarizing Customer Reviews. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 168-177). ACM.

14. Liu, B. (2012). Sentiment Analysis and Opinion Mining. San Rafael: Morgan & Claypool.

15. LSTM Networks for Sentiment Analysis. DeepLearning 0.1 Documentation. Retrieved December 01, 2017, from http://deeplearning.net/tutorial/lstm.html.

16. Mitchell, J. (Datasnaek). "Trending YouTube Video Statistics and Comments." Kaggle, Kaggle Inc., Aug./Sep. 2017. https://www.kaggle.com/datasnaek/youtube/

17. Moraes, R., Valiati, J. F., & Neto, W. P. G. (2013). Document-level Sentiment Classification: An Empirical Comparison between SVM and ANN. Expert Systems with Applications, 40(2), 621-633.

18. Pozzi, F. A. (2017). Sentiment Analysis in Social Networks. Amsterdam: Elsevier.

19. Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - EMNLP 02.

20. Turney, P. D. (2002, July). Thumbs Up or Thumbs Down?: Semantic Orientation Applied to Unsupervised Classification of Reviews. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (pp. 417-424). Association for Computational Linguistics.

21. Wang, X., Liu, Y., Sun, C., Wang, B., & Wang, X. (2015b). Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 1343–1353, Beijing, China. Association for Computational Linguistics.

22. Whitelaw, C., Garg, N., & Argamon, S. (2005, October). Using Appraisal Groups for Sentiment Analysis. In Proceedings of the 14th ACM International Conference on Information and Knowledge Management (pp. 625-631). ACM.

5 Appendix

1. Chain rule for repeated applications of conditional probability.

$$\begin{aligned} p(C_k, x_1, ..., x_n) &= p(x_1, ..., x_n, C_k) \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2, ..., x_n, C_k) \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)p(x_3, ..., x_n, C_k) \\ &= ... \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)...p(x_{n-1}|x_n, C_k)p(x_n|C_k)p(C_k) \end{aligned}$$

2. Naive Assumption of conditional independence to simplify model. This the joint model can be derived via:

$$\begin{aligned} p(X_k|x_1, ..., x_n) &\alpha p(C_k, x_1, ..., x_n) \\ &\alpha p(C_k)p(x_1|C_k)p(x_2|C_k)p(x_3|C_k)... \\ &\alpha p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \end{aligned}$$