Blind Tweet Classification and Clustering in Arabic

McClain Thiel
ME315 LSE Summer School
7-4-2019

McClain Thiel
5-7-19
Machine learning in practice

Blind Tweet Classification and Clustering in Arabic

The goal of this project is to use the data to build a system that is able to classify the sentiment of a tweet as either positive or negative and to use unsupervised learning to group tweets into groups with similar themes. The project is pretty typical of text-based data sets but with a twist. The tweets are written in Arabic of which I do not speak. This presented an interesting challenge to many aspects of the project that made for a more satisfying and generalizable result.

The Data

The dataset is a collection from the UCI machine learning repository consisting of 1000 positive and 1000 negative tweets. The data was classified as one of the two options by a native Arabic speaker. All tweets are written in either modern standard Arabic (MSA) or the Jordanian dialect but the are not labeled as to which. Characters consist of Arabic script characters, English characters, and twitter emoticon character.

Preprocessing

The first task is cleaning and organizing the data in such a way that it can actually be fed into the models. Regularizing the data involves several tasks including cleaning, stemming, tokenization, and mapping. For these tasks I used popular libraries to build a collection of data that was compatible with models. First, I cleaned the data by removing all punctuation and making the whole tweet in lowercase. This helps with computing comparisons. If the order is to find the sentiment of a tweet, the system needs to understand that “The Cat, and, the dog!!” is referring to the same thing as “The cat and the dog!”, and the easiest way to accomplish this is to regularize all the formatting to something like “the cat and the dog”.

The next step in the process is called stemming. It’s another way to improve computational comparisons between words. The process involves stripping all words down to the root so that the meaning is more apparent. For example, the words “sit”, “sat”, and

"sitting" would all be transformed to "sit" because they all mean the same thing in some respect. The danger of this technique is that it removes too much information. If the stemming process removes a negation like "un" or "dis" in English, the meaning of the word is actually reversed. However, even with this caveat, the process of stemming is still widely practiced as it almost always gives more than it takes. I accomplished this by finding a database of Arabic roots and comparing my words to the database and stripping the prefixes and suffixes until the words matched.

After stemming the next step is removing what are called stop words. Words like "I", "they", "am", "be" and so on don't really tell us anything about whether the text is negative or positive, but due to their abundance, they do significantly dilute the potency of more relevant word. It's important that generic words like pronouns and articles are removed from text when building a classification model. I managed to remove these words by finding another Arabic database and just running a comparison test.

Finally, I had to tokenize and map the words. Tokenization is the process of converting a sentence to a list of words for computational purposes. Generally, this just involves splitting the text on the whitespaces and moving each individual item into an array. Mapping just involves representing each word as a number. Machine learning models are rarely built to process text and even if they are, many computations are faster between numbers as opposed to words. So, to map the words I simply made a list of every unique word in the database and then assigned an arbitrary number to them. In this case I just used sequential numbers. This allowed me to encode and decode tweets to and from series of integers and word with more complex models.

Classification

The first method I attempted to use was a naïve Bayes classifier. This basically works by finding how often words are used in positive tweets and how often words are used in negative tweets and building a database of the relative frequencies of every word in the training set.

$$Relitive\ Frequency_{Positive} = \frac{\#\ of\ occurrences\ in\ positive\ tweets}{total\ number\ of\ words\ in\ database}$$

I looped through every word of every message and counted how many times each word appeared and in what context and then build a table of relative frequencies for every word appearing in a positive tweet and every word appearing in a negative tweet. From here I was able to build the model.

The model is based on Bayes theorem which says that the probability of one event given the occurrence of another is equal to the probability of the intersection of the two events divided by the probability of the given event.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

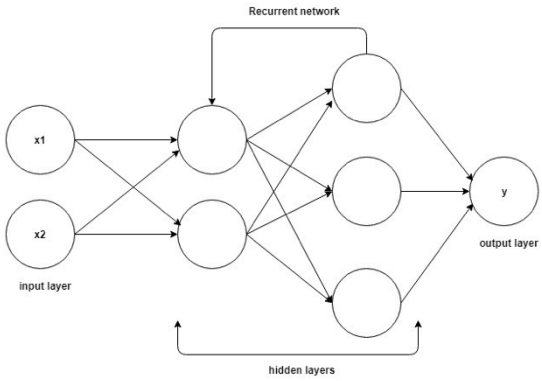
If both words and positive vs negative sentiment are viewed as random events, Bayes' rule can be used to classify tweets. Let event A be positive or negative, and event B be every word of the tweet. The calculation becomes $P(positive|tweet) > P(\text{negative}|\text{tweet})$. If this is true then the tweet is classified as positive, else negative. Because I am using the relative frequency of words and not tweets the actual equation is the sum of the probabilities as show above of all the words of a given tweet.

$$\prod_{i=words}^{tweet} P(positive|word_i) > \prod_{i=words}^{tweet} P(\text{negative}|word_i)$$

One problem with this method is that it only words as shown above if every word in the tweet is present in both the positive word database and the negative word database. This problem, however, is easily circumnavigated by simply adding a constant to the sum where the word exists in one database but not the other. The main problem with this method is that its only effective in very large datasets with more polarized words. This data set was relatively small and pretty basic. The most common words were all pronouns and articles the words that weren't didn't tend to convey particularly strong emotion. Due to these factors, this method produces very modest but consistent results. The model successively classified between 58% – 60% of samples correctly on an 80/20 train test split multiple times. This is only slightly better than just guessing randomly whether a tweet is positive or negative.

I also attempted to use an n-gram modification to this process but due to the fact that these items are tweets, and therefore often very short, the n-gram proved both difficult to implement and provided a negligible improvement in accuracy. I also implemented a term frequency inverse document frequency version of the model but after I removed the stop words, and grouped the single occurrence words, the database was both small and diverse enough to where the TF-IDF didn't make much of a difference. It's not uncommon to have tweets that have less words than are in an n-gram model. If I chose to use grouping of three words, but the tweet was just two words and a few symbols or emojis, the n-gram model is not only completely ineffective, but difficult to implement due to the fact that the system won't be able to easily handle the exception. Due to these data limitations I decided that the best course of action was to use a very simple naïve Bayes classifier and then redistribute efforts into building a more complex and hopefully accurate model with a better chance of success.

The next model I decided to build was a recurrent neural net (RNN). RNN work like normal neural nets except that there is a layer that feeds back into itself. This allows RNN's to model temporal behavior. This works very well for text processing and analysis because the order of words can be thought of as a temporal process. The net will adjust to words like "not" and "no" in front of other words because it's able to store the negation in one cycle and see how the negation affects the next word after its fed through the recurrent layer. It allows the model to develop some sense of context and word relation.



The math for neural networks in general but recurrent NN in particular is very complex and largely out of scope for this paper but I will briefly touch on how cost and activation functions differ in RNN vs feed forward NN. RNN's calculate the cost function as a sum of a series of occurrences, in this case words, and then adjusts weights to minimize cost. A rough model of a cost function for a model like this might look like:

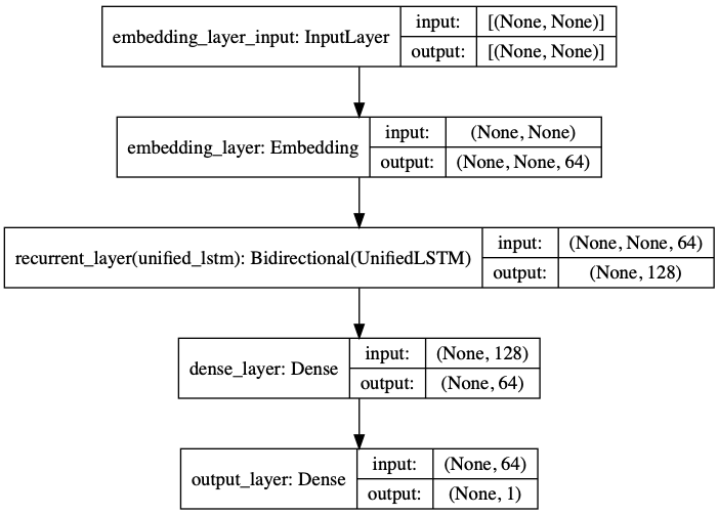
$$Cost = \frac{1}{cycles} \sum_{t=1}^{cycle} \sum_{j=1}^{words} activation(t, j)$$

This is showing that the cost is calculated as a series of the previous several time series items, in this case words, as opposed to just the previous weight. This is why the net is able to understand patterns in series of items such as words in sentences and why it is such an effective tool for sentiment analysis and text processing.

Specifically, all the models I used were long-short term memory systems (LSTM). The recurrent layer of every model is actually a LSTM layer which is capable or remembering learned behavior over an arbitrary time. This is accomplished by use of nodes with three gates, the input gate, the output gate and the forget gate. These nodes determine how and in what ways the input features interact via learning weights through training, and then retain this information in perpetuity. This is how a RNN may learn the concept of negations or pronouns in a way. If the node say an input it had previously identified as positive with the addition of the word 'not' or 'no', and then checked with the target value and discovered that the result was actually the opposite of what it expected, the net would begin to learn that the added word would reverse the meaning when placed in front of other words and then be able to generalize

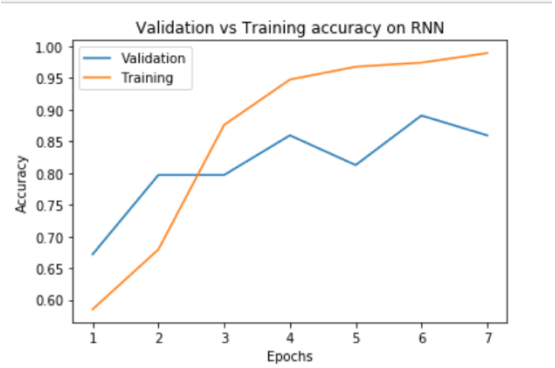
this principal to other examples. This is why RNNs are particularly well suited to anything that can be modeled as a time series such as a constant stream of text.

The best model I used consisted of 5 layers. The first two layers are embedding layers followed by a bidirectional recurrent layer and then finally two fully connected dense layers.



The first layers act primarily as input and embedding layers, while the recurrent and first dense layer do many of the computations. The last layer is just an output layer which reports the final classification. This model used the encoded versions of the words represented as an integer array to keep everything clean. The RNN basically just acts as a complex interpolation machine at a low level so all that is really happening is that a series of numbers are being mapped to positive and another series of words are more closely associated with negative. However, in this case, this is a very effective method to map tweets to sentiment.

This model achieved 89.7% accurate classifications on a test set of 400 tweets. This is a very respectable percentage given the size and nature of the data set and my lack of ability to fine tune anything due to the language barrier. The maximum length of a tweet is 140 character and so the RNN's ability to learn from context is somewhat hobbled by the short nature of the texts. Getting about 90% accuracy form such a sub optimal data set is pretty good and significantly better than the ~59% of the naïve Bayes classifier. Although in spite of its abysmally low accuracy, the NBC does have some advantages over the RNN.



The fact that it is not a black box system means that we can look at its innerworkings and uncover relevant information about the underlying data. One of the most helpful and interesting insights is which words are most relevant to the classification as either positive or negative. We can reverse the classifier and find out which words are most likely to get a tweet classified as negative or positive. The inverse relative frequency should give a general idea of the feeling of the people in the data set about the topics they tweet about. If one word appears

very often in the positive tweets but is very rare or absent from the negative tweets, this may tell us that the people in this database feel very positively about this subject. This isn't necessarily true, but the presence of this word does reveal something. Of words that appeared in both positive and negative tweets, the following table of words were the most helpful in determining if a tweet was positive or negative:

The naïve Bayes classifier suggests that, of words that show up in both the positive and negative databases these are the words that are most influential in deciding how a tweet is classified, but possibly more telling is the list of words that only appear in one database or the other. For example, below is the table of words that were only mentioned in a negative context and their translations.

Arabic	مفاعل	نووي	طاق	اردن	احن	عاه	بلد	يلع	شعب	دول
English	Reactor	Nuclear	to be [+accusative]	Jordan	I love you	Disabled	Country	Oh shit	People	Ugh

Arabic	خير	رب	ارب	مسلم	برنامج	رائع	جميل	صالح	سلم	جعل
English	Good	God	Arb	Muslim	Program	Wonderful	Nice	Valid	Peace	To become

This seems to give a much more specific and insightful account of the data. While the words that are mathematically the most relevant are words that one might associate with positivity, they are relatively generic and don't really add anything specific to the data. These words show that the group that this data was pulled from has strong feelings on very specific issues. Most notably, it would seem based on this data that a specific nuclear reactor, or nuclear reactors in general are not very popular, and that people maybe upset with the country of Jordan. It turns out that when this data was collected in 2014, nuclear power in Jordan was a hot political topic. The state of Jordan has been investing in nuclear power for many years but in 2013 Greenpeace Jordan released a report called "Future Energy of Jordan" which condemned the country's nuclear program due to seismic and water concerns. This is likely the cause of this trend in the data but there is not was to draw a definitive conclusion based on the available data.

Similarly, this same type of analysis can be run on the positive words. However, while the data was very insightful for the list of negative words, the list of positive words, is a prime example of why this is not always helpful. All of the words appear to be very generic and not reveal anything specific about the population.

Arabic	ربن	صباح	رزق	حمد	بار	خلق	نور	صلح	رضي	قلوب
English	hearts	Satisfied	Peace	the light	Create	Bar	He praised	Livelihood	Morning	To raise

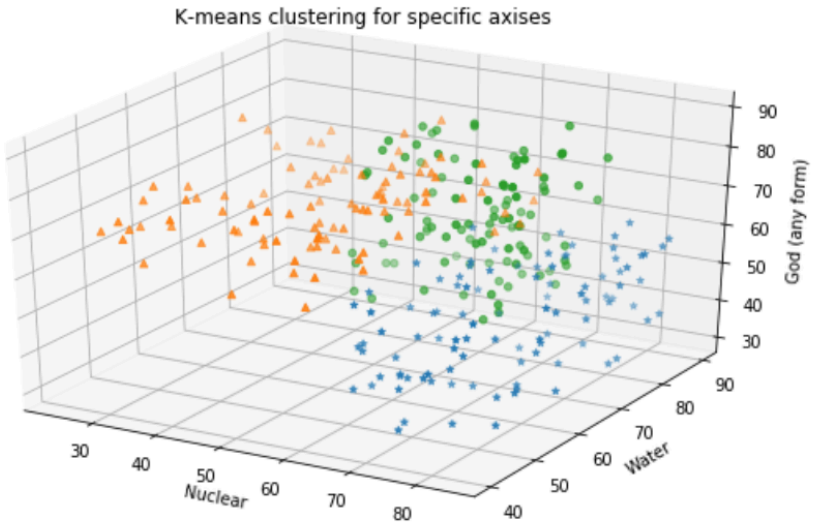
This is the list of words that appear exclusively in a positive context within the data set. These words are not nearly as helpful as the negative words. It best they may suggest that many of the positive tweets were tweets wishing a good morning or some sort of religious / inspirational message. At worst, it may be that these words have nothing in common and this data is useless.

Clustering

The next phase of this project was to use unsupervised learning techniques to find patterns in the data the reveal something about the population the data set was drawn from. The first technique I used to accomplish this was K-means clustering via common word usage. The process of K-means clustering is pretty simple. It just involves grouping the data into k groups by their proximity to each other as judged by Euclidian distance. The slightly more challenging aspect is expressing the 'location' of a tweet in a way to make this calculation possible. I did this by creating a matrix of all the possible unique words and used that as the auxesis and then plotted the corresponding value as the number of times that particular word occurs in that tweet. The result is a sparse matrix representing all of the words and how many times the words were used in a tweet.

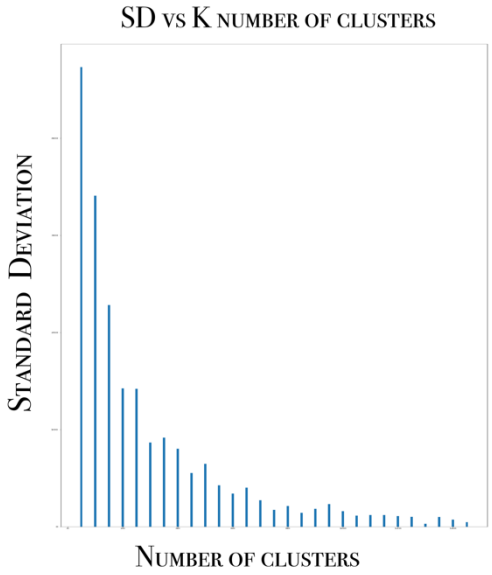
$$distance = \sqrt{\sum_{words}(\#\ of\ occurances\ in\ tweet\ 1 - \ \#\ of\ occurances\ in\ tweet\ 2)^2}$$

The most challenging part of K-means clustering if figuring out what the value of k should be or how many groups to separate the data into. Because there's isn't really a mathematically definable goal to the clustering, it's hard to determine computationally when the groupings are optimal and much of the work is left up to manual interpretation. To the right is a simulation of what k-means clustering might look like if the dimensions were reduced from thousands to just three and the data where regularized. Each new type of point would represent a new cluster with tweets with many words in common being grouped together. Even with its



limitations, K-means clustering is still highly effective at grouping data in interesting and insightful ways. I first ran the program with 5 groups. One group consisted of only three tweets and upon further inspection, I found that all three referenced the nuclear reactor in Jordan and cited lack of available water as their principal concern. Again, with k equals 5 groupings, another group reported another specific problem. Twenty-six tweets were grouped together condemning the nuclear powerplant but this time their main concern was something other than the lack of water. Many cite financial concerns; governmental incompetence is another common concern, but they all have the same general message. K-means clustering revealed in far more specific terms what naïve Bayes had hinted at. From the data alone, the machine learning algorithms were effectively able to find a political issue and judge public opinion and group the data into specific reasons for the general displeasure.

One issue with K-means clustering is that it can create wide disparities in the size of the groups, so if, for example, one ran K-Means with 50 groups, the program could and often would create on group with over half the data points and many groups with only one or two data points. This sort of distribution doesn't really tell anything about the data. A single data point doesn't gain any information in a cluster of its own, and if a single cluster has too much of the data its similar to looking at the entirety of the data. In order to find numbers of clusters that produced informative results, I looped through possible values of k and calculated the standard deviation of the sizes of the grouping. Once I found values with relatively small standard deviation, I manually inspected the content of the groupings for patterns. The plot of the standard deviation in relation to the number of clusters shows a somewhat asymptotic but also randomized relationship. Because clustering is more freeform and mostly involves finding patterns in the data, the next step is usually to see if I can draw any conclusions about the population from the data. Once I form a hypothesis, I will look into the issue and see if there Is any additional information available on the internet that corroborate or refutes my hypothesis.



From my analysis, I have concluded that the data can largely be grouped into three main categories: political, religious, and twitter. The political category is relatively small compared to the other two. It has many subdivisions, almost all of which take a stance on the nuclear power

plan. The second group is religious centered text, and while its larger than the first group its still significantly smaller than the third. This group consists of generic blessings, prayers and Quran passages. The third and largest group is what I will call twitter. This group consists of messages that exist and make sense solely in the context of twitter. Some example of tweets like these and their translations are below.

Text	احسنوا الظن#	احسن علاج هذا	انا احب هذا الكاتب	جميل جدا	صح جدا
translation	# Think better	This is the best cure	I love this writer	Very beautiful	Very true

These tweets don't really mean anything outside the context of twitter, but on the website things like this often make sense and have meaning. I speculate that many of the above texts had some other media, like a link or picture, attached to the original post. They may also have been replies to other tweets which are often stored as the same type of data. Hashtags, associated with the # sign are a common twitter tool to show users how a post relates to other posts, but users often use the hashtag alone to show they support the thing represented by the tag. Almost half of the tweets in this data set would be categorized as something like this. Many are not full sentences or even words, many tweets consist of symbols, emojis, or some other non-script-based form of communication. Beyond these 3 categories there are numerous examples of tweets that evade classification by me or the program. Examples of this include a two-paragraph rant about a gas station, a man complaining about the necessity of morning showers, and various common complains pertaining to economic instability, American imperialism and any number of other things.

The clustering analysis is far from perfect. It has trouble grouping short tweets with similar sentiments but different words. Synonyms often pose a challenge. For example, if one tweet read "My son attends university" and another read "My boy goes to college" the system wouldn't recognize that the sentiment was basically the same because the only word in common is 'my'. This problem is largely negated in longer tweets in categories with more samples effectively due to the transitive property, but short strings are often grouped poorly. Another issue is the presence of many similar words in unrelated tweets. Basically, the inverse of the first problem.

Although imperfect the clustering analysis provided a significant amount of insight albeit in a somewhat subjecting and manually intensive way. By iterating through different values of k I was able to see how tweets were grouped and then subdivided again and again. No one value of k provides optimal groupings, because if k is too large relative to one group, that groups subdivisions are too fractured, while simultaneously, other groups subdivisions may be far too course. In the future this may be a good use of some sort of hierarchal clustering analysis

Conclusion

Through machine learning techniques, I was able to build a reasonably successful classifier in a forging language and discover contemporary political and cultural trends through a twitter database. The practice of sentiment analysis has already been well studied in English but has largely been neglected in relation to many other languages, especially those that don't use the Latin script, but a large portion of the world speaks a language other than English and much of the world data available is written in not Latin scripts. This project aims in part to show that many of the sentiment analysis techniques used in English are still effective in other languages with minor modifications. Some of the techniques are generalizable to symbolic or non-character based 'text', which may prove useful when trying to apply machine learning techniques new novel domains. For example, once the root meaning of the words is determined and mapped to the database, the language in which the original text is written is irrelevant to the classification or regression task. This means that anything that can be hashed to a concept and a unique identifier can be used to train a recurrent neural net and then predict or classify new data. Sentiment analysis techniques can therefore be applied to symbol based written languages. These techniques can be applied to east Asian languages such as Chinese, Japanese, and Korean, in addition to pictographic languages like ancient Egyptian hieroglyphics and Aztec languages.

The breadth of possible applications is enormous. Advanced sentiment analysis may prove more effective than pure translation in many situations and will certainly eliminate many of the hiccups of natural languages. The symbolic nature of the program may also be generalizable to non-languages and other special cases. RNN's structure fit naturally with the flow of programming, and it's conceivable that a system could be made that would be able to classify classes, functions and other substructures of program by intent or use and label or even predict other functions. This could shift the programming paradigm from functional to simply declarative in many cases, or at least large-scale abstraction would be much cleaner and faster. A programmer could describe what he or she wanted the program to do and a regressive or generative RNN would be able to use past knowledge of programs to suggest or deign a new program to the specifications of the programmer.

Symbolic sentiment analysis via recurrent neural nets with subpar resources and non-optimized procedures proved reasonably effective at classifying and clustering a foreign language. From the classification I was able to predict whether a tweet would be considered positive or negative by a native speaker and via clustering I uncovered a heated political trend and noticed social/cultural trends that I would not have otherwise know about. With further technique refinement and access to more data and time I'm sure the results would be even more promising. I believe that RNNs will prove highly effective at a wide variety of essential tasks in the near future and have massive potential to make interesting discoveries and bring about positive change.

Bibliography:

/@suvro.banerjee16. “An Introduction to Recurrent Neural Networks - Explore Artificial Intelligence.” *Medium*, Explore Artificial Intelligence, 17 Sept. 2018, medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912.

GHATAK, ABHIJIT. *DEEP LEARNING WITH R*. SPRINGER, 2019.

Hastie, Trevor, et al. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2017.

James, Gareth. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2017.

“K-Means Clustering in R Tutorial.” *DataCamp Community*, www.datacamp.com/community/tutorials/k-means-clustering-r.

Li, Susan. “A Beginner's Guide on Sentiment Analysis with RNN.” *Towards Data Science*, Towards Data Science, 2 June 2018, towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e.

“Recurrent Neural Network.” *Wikipedia*, Wikimedia Foundation, 1 July 2019, en.wikipedia.org/wiki/Recurrent_neural_network.

Silge, Julia, and David Robinson. “Text Mining with R.” *2 Sentiment Analysis with Tidy Data*, 16 May 2019, www.tidytextmining.com/sentiment.html.

Github link to repo: https://github.com/McClain-Thiel/sentiment-analysis-in-arabic