React

Derek Huang, Brandon Truong, Eli Pugh

June 12, 2018

https://github.com/derekahuang/React

Introduction

Social media, namely Facebook, had come to be at the forefront of modern communication, politics, advertising, and media consumption. It's an integral part of how we interact with others – today, support is not shown through letters and flowers, but rather through likes and comments. Because there is so much content, however, it is difficult for a person to see content they are both interested in and in the mood for. Because of increasingly polarized audiences, it is important to understand how audiences will receive the media you post and how you word it.

By understanding the kind of reactions posts would garner, we can not only augment our sentiments towards posts (i. e. help us respond to posts automatically), but also be able to filter content (regardless of the actual reactions they have been given, since many posts on Facebook have sparse data reactions) based on the type or mood they are. That way people can filter the content they actually want from the content they don't want depending on their mood. Users would also be able to use this as a metric to determine posts with optimal semantics for the messages they are trying to convey. This would particularly be useful in the context of news on Facebook, where filtering and prediction would be very useful for both publishers and consumers to post and consume content intelligently.

A number of difficulties are involved in this problem. The polarized and varying audiences react distinctly for different news sources. Certain words, entities (celebrities, groups), etc. garner different sentiment over time, and a shift in general perception of them can change rapidly. Most posts on Facebook also have sparse reaction data, which don't provide much useful information on how people would react to it.

Related Work

Sentiment classification is a topic of huge interest. Its natural extension to Facebook statuses has, however, been relatively sparse. There have been several papers and blog posts written about classifying Facebook statuses into various sentiments. Akaichi et al. published a paper at the 17th International Conference on System Theory, Control and Computing (ICSTCC) on "Text mining Facebook status updates for sentiment classification" where they explored the effects of various n-grams featurizers paired with Naive Bayes and SVM on classification accuracy. To do this, they built their own sentiment lexicon based on the emoticons, interjections and acronyms, from extracted statuses updates. Their results achieved at most 72% classification accuracy, and any features besides unigrams decreased accuracy.

A more notable blog post was published by Max Woolf in 2016, where he used a pretrained recurrent neural network, which allowed it to handle text at the character level, allowing it to incorporate capitalization, grammar, text length, and sarcasm in its predictions. Woolf acknowledges that reaction data can be used to classify sentiment, but "only if there is a large amount of data available," which he did not have access to at the time of his experiment (2016).

We thus come to this problem with a fresh perspective by analyzing news posts on public Facebook pages. We had hoped that by scraping 16,000 posts, we would accumulate enough data to circumvent the problem Woolf faced. Also, by introducing various featurizers such as Word2Vec, FastTest, and GloVe, we hoped to add a layer of complexity over the strictly n-gram features of Akaichi et al.

Task Definition

The problem may be defined as a multiclass-classification task. Our system will take in a news post from Facebook (abstractactable to a body of text) and output a vector of proportional weights representing the predicted reaction spread for the post. This would look like:

Input post: "5 years ago today, 20 first graders and six faculty were lost during a mass shooting at Sandy Hook Elementary School. Here's how the Sandy Hook victims live on in the hearts and the work of their families:"

Output: [.077, .004, .002, .911, .006] (loves, wows, hahas, sads, angrys)

This output can then be used to generate a set of reaction-amounts for a given post sample. Based on this, for existing posts, you can filter out ones of certain proportions of reacts to match your mood. For new posts, you can see an estimate of how it will perform and be received.

Approach

Baseline:

Our baseline is simple self-written linear regression. We first strip the text of punctuation, make it lowercase, and split examples to train and test. We also turn the reactions into percentages, so that the components of each example vector sum to 1. The features for each post are unigrams and bigrams, although we also experimented with sliding window character features for comparison. We're optimizing mean squared error using Stochastic Gradient Descent.

Proposed Methods:

- The first step, data aggregation will be scraping Facebook Statuses through the Facebook API and other methods. We are essentially aggregating a large list of Facebook Page IDs and pulling all of the post, post-link and reaction data from each of them.
- For our learning algorithm we are using a multi-layer perceptron (MLP). We chose this as algorithm because of its capabilities to learn non-linear models and its relative simplicity as a neural network. Out input layer is composed of n nodes labeled $\{x_1 \dots x_n\}$ where each node represents a feature. We have t hidden layers, where each hidden layer can have any number of nodes. For each node, we have a set of weights $\{w_1 \dots w_n\}$ and we compute the weighted summation $\sum_{i=1}^{n} x_i w_i$ and then pass it into an activation function. We are experimenting with logistic: $g(z) = \frac{1}{1+e^{-z}}$ as well as hyperbolic tan $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ to determine the value for that node. This value gets passed forward until the output layer, where we take the max of each node and 0, and then normalize into a stochastic value vector.

For loss we are using squared loss, which we define as

$$Loss(y, \hat{y}, W) = \frac{1}{2}||\hat{y} - y||_2^2 + \frac{\alpha}{2}||W||_2^2,$$

where $\frac{\alpha}{2}||W||_2^2$ is used to penalize complex models and α is a hyperparameter to tune our penalty. With these we seek to prevent overfitting. We backpropagate to determine the weights of the hidden layers using Adam, which stores an exponentially decaying average of past squared gradients to adjust the learning rate and an exponentially decaying average of past gradients to keep momentum and escape local minima.

- As per our mentor suggestions, we plan to experiment with different types of featurizers and evaluate which ones work best with a multi-layer perceptron. In addition to experimenting with features, we plan to experiment with the number of hidden layers, the number of neurons in each hidden layer, etc. Our first featurizer is word unigram and bigram. Many words, such as "a" and "the" are frequent but have low importance. We thus chose to represent each unigram and bigram as its TF-IDF weight. For a given term t, we calculate

$$TF(t) = \frac{\text{\#of times term } t \text{ appears in a status}}{\text{total \# of terms in the status}}$$

and

$$IDF(t) = \log_e\left(\frac{\text{total \# of statuses}}{\text{\# of statuses with the term } t \text{ in it}}\right)$$

.

We then normalize by the Euclidean norm, and use the $TF(t) * IDF(t)$ as the weight for the feature.

- In addition, we plan to experiment with Word2Vec, Fast Text, and GloVe. Word2Vec utilizes context to construct a 3-layer neural network that predicts some task, where the hidden layer encodes word representations. We can do this two ways. The first option is a Continuous Bag of Words (CBOW) model, where given some context $w_{i-1}, w_{i-1}, w_{i+1}, w_{i+2}$, we seek to predict word w_i. The second option is skip-gram, where given a word w_i, we seek to predict its context $w_{i-1}, w_{i-1}, w_{i+1}, w_{i+2}$. There are many nuances to play around with, such as random sampling of window sizes as well as negative sampling, where we update the weights for positive label words as well as a certain number of negative label words. The hidden layer matrix is a |word count| x |hidden layer node count| matrix that is equivalent to a |word count| x |feature count| matrix. Each word is represented by |feature count| features in the hidden layer, and we can extract these features to use for another model.

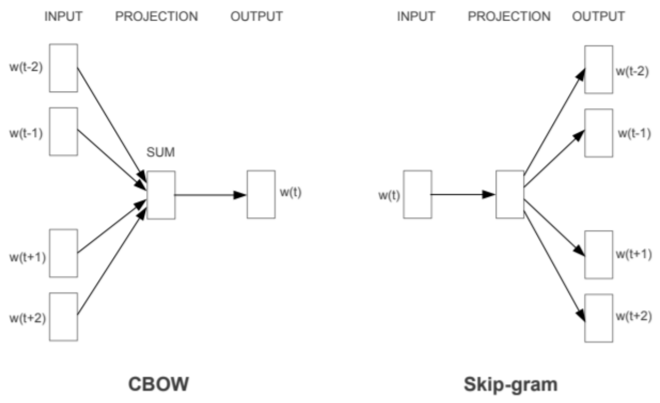


Figure 1: Photo taken from Mikolov et al., 2013 at https://arxiv.org/pdf/1301.3781.pdf

FastText is seen as an extension to Word2Vec that breaks words into n-grams. For instance, the word "water" may be broken into several tri-grams "wat," "ate," and "ter." The word embedding thus becomes the sum of these n-grams. This helps with rare word representation as it is highly likely that some of their n-grams appear in other words.

Finally, GloVe is a count-based model that builds a word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus. It seeks to learn word vectors such that the dot product between vectors equals the logarithm of the probability of co-occurrence. The error function is defined as $\sum_{i,j=1}^{V} f(X_{ij})(w_i^T\tilde{w}_j + b_i + \tilde{b}_j \log X_{ij})^2$ where the model seeks to learn w, b and X_{ij} is the co-occurrence probability. Because the logarithm of ratios of co-occurrence probabilities is equal to the logarithm of differences, this model seeks to associate logarithmic differences with vector differences in the word embeddings.

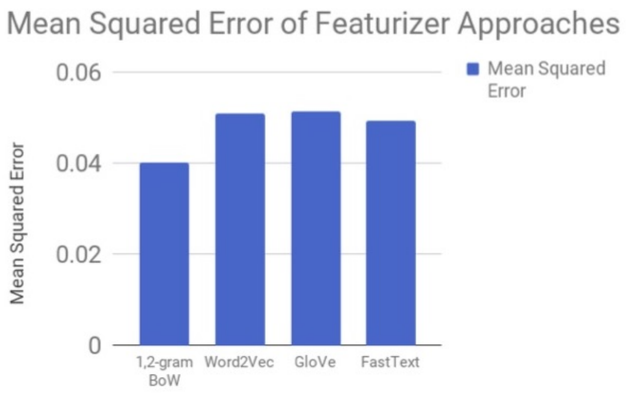
- After some more thought, and some more classes, we hope to use more advanced techniques for NLP. In particular, we hope to incorporate state-of-the-art Neural Machine Translation techniques that involve sequence-to-sequence encoder decoder model with an attention mechanism. As specified in Hassan et al. in "Achieving Human Parity on Automatic Chinese to English News Translation," attentional sequence-to-sequence NMT models the conditional probability $\mathbf{p}(y|x)$ of the translated sequence y given an input sequence x. This has been shown to work extremely well due to the large amount of parallel text data encoding obtained via machine translation and has been shown to work well with complex NLP tasks, most notably obtaining parity in Chinese-English translation.

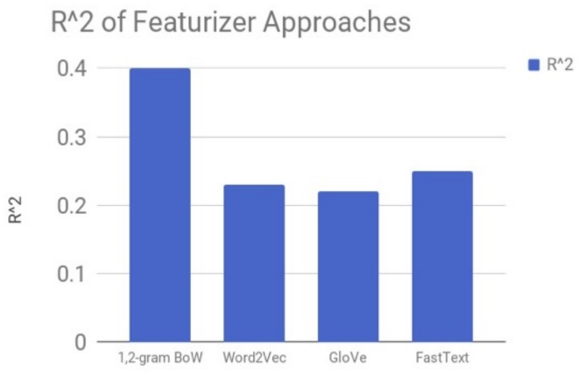
Results

For our baseline, on a sample run with 500 epochs of SGD, we have a mean squared error of 0.2432 on the test set.

For our current pass through MLP with TF-IDF unigram and bigram weights with one hidden layer of 100 neurons, a logistic hidden layer activation function and using Adam optimizer, we have a mean squared error of 0.04088 with R^2 score of 0.39, which represents the coefficient of determination. It is measure of how well future samples are likely to be predicted by the model.

Our Word2Vec, FastText, and GloVe had similarly small mean squared error but less impressive R^2 scores. All 3 featurizers had error between 0.049 and 0.052, and R^2 between 0.22 and 0.25.



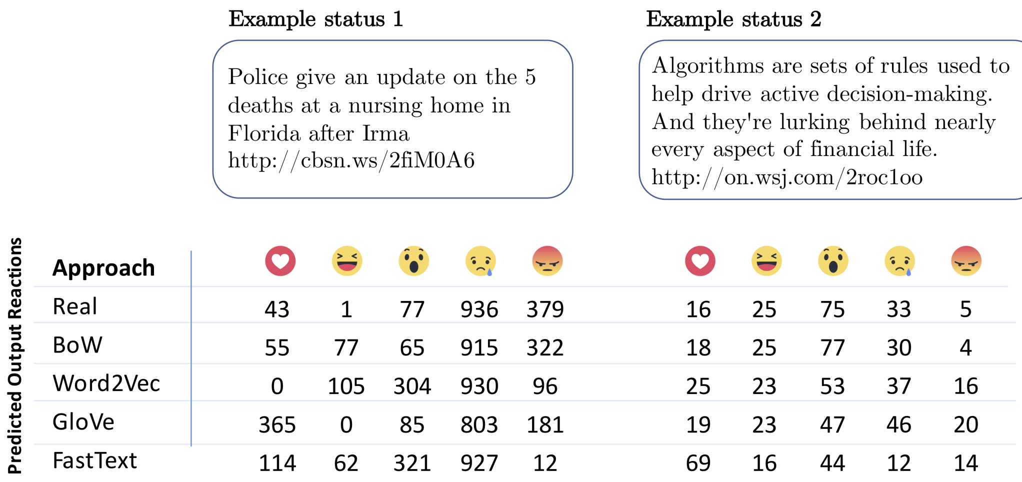


Metric:

- Our output is our relative strength of each reaction normalized into a unit vector. We will compare our output to the normalized proportion of reactions of a status. Our cost metric is the sum of squared distance for each of the five reactions, or more formally as in the Proposed Methods section:

$$Loss(y, \hat{y}, W) = \frac{1}{2}||\hat{y} - y||_2^2 + \frac{\alpha}{2}||W||_2^2.$$

- The baseline does not perform wonderfully. Our dataset is diverse and complex enough that simple linear regression on unigrams and bigrams doesn't generalize well to unseen examples. With an average loss of 0.2432 on the test set, we hope to improve nearly 10-fold with more advanced methods.
- Our metric is designed to indicate if the model did well in predicting what proportions of people will react in each way to a given post. Statuses often have a range of reactions, with some having similar numbers for several different reactions. Because of this, we chose a metric that will capture overall accuracy of each proportion, rather than just a classification that predicts the most likely reaction. This metric allows us to capture in more detail the true sentiment of the post.
- With our results thus far, the MLP performs noticeably better than the baseline. We have currently only tested on MLP with unigrams and bigrams with TF-IDF weighting. We expect to improve on this even more with Word2Vec, Fast Text, and GloVe.



Example status 1

Police give an update on the 5 deaths at a nursing home in Florida after Irma http://cbsn.ws/2fiM0A6

Example status 2

Algorithms are sets of rules used to help drive active decision-making. And they're lurking behind nearly every aspect of financial life. http://on.wsj.com/2roc1oo

Predicted Output Reactions

Approach	❤️	😆	😮	😢	😠	❤️	😆	😮	😢	😠
Real	43	1	77	936	379	16	25	75	33	5
BoW	55	77	65	915	322	18	25	77	30	4
Word2Vec	0	105	304	930	96	25	23	53	37	16
GloVe	365	0	85	803	181	19	23	47	46	20
FastText	114	62	321	927	12	69	16	44	12	14

Figure 2: Example predicted reaction distributions

Data:

In order to develop our baseline and try out some simple featurizers with MLP, we scraped 16,000 posts from top-20 news Facebook pages (i.e. CNN, Fox, BBC). Currently evaluation is done on this dataset with a random 80% used for training and 20% for testing. We may scrape more to test with later, as a larger corpus will likely help in predicting more diverse status inputs.

Quantitative Analysis:

We achieved the best results with a unigram and bigram featurizer using TF-IDF. There are several reasons why MLP with our simplest featurizer returned the best results.

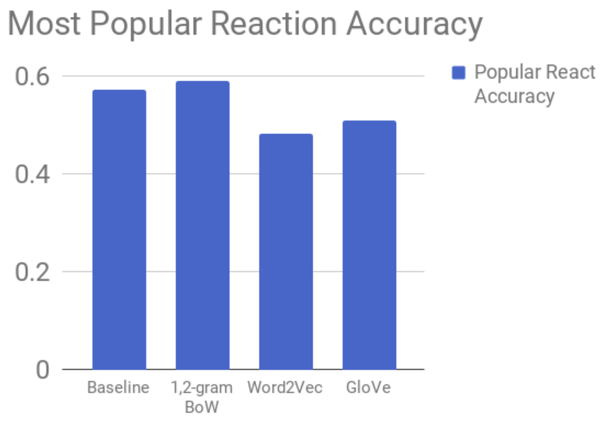
- We did not prune our data of low reaction statuses. Thus, we left in Facebook statuses with just one or two reactions that would have a extremely high proportion of its reactions in one category due to the nature of a low reaction count. However, our algorithm weighted these identically, potentially skewing our data.

- We scraped our data from political websites, including Brietbart and Washington Post. These websites are very far learning, and although they may post the same status, the reactions of their audiences will be very different. Thus the political nature of our statuses makes it very hard to predict how people will react.
- A multi-layer perceptron is a relatively simple model, whereas Word2Vec, FastText, and GloVe are state-of-the-art featurizers. It is possible that MLP does not handle these complex featurizers very well, working better with the simpler features.
- Many of our errors also came from not understanding context. Using MLP was definitely the bottleneck of our model. An LSTM might remove this bottleneck and allow the model to use context to catch current errors.
- For Word2Vec and FastText, the corpus built from our data was too small to form meaningful relationships. We only had 16,000 statuses, too few to generate a comprehensive corpus.
- Our GloVe model uses pretrained word vectors that have at most dimensions of 300. This means that each sentence is represented by 300 features. Our Tfidfvectorizer is set to use as many features are there are unigrams and bigrams, which far exceeds 300. GloVe could perform worse purely on the fact that it has nowhere near the same number of features as unigram and bigram TFIDF.
- Within Word2Vec, we found that skip-gram worked the best for us. With skip-gram, we are trying to predict the context of a word given word w, whereas with CBOW, we are trying to predict word w given its context. It makes sense that skip-gram works better for semantic analysis, as we are trying to predict the reaction of the entire status given the word, and therefore would do better trying to predict the context.

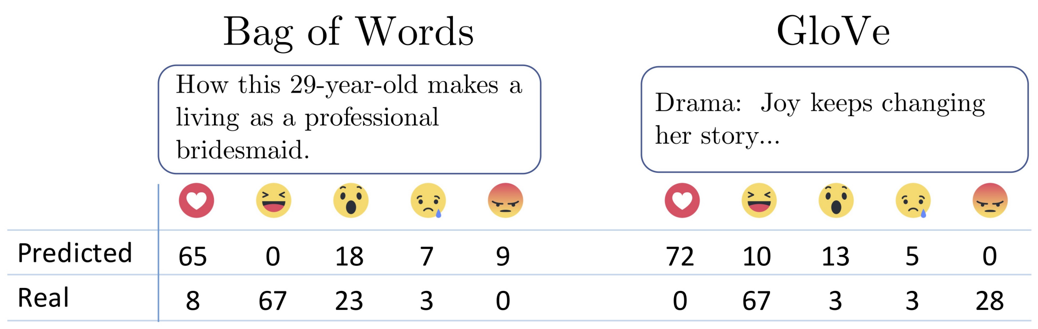
Qualitative Analysis:

While testing different models, we have experimented with example posts and checked to make sure that our predictions were reasonable to human observers. On sample inputs, our models seem to be fairly sensible, though less though on more specific and nuanced text that may have some words not in the training data.

One qualitative check we ran was testing what percent of the time our algorithm predicted the most popular reaction correctly. Though our task was regression and not classification, this checks that our model is at least understanding the most obvious sentiment of each post.



We also printed and read some of the statuses with the highest error, and noticed some common themes. Below we have included two example statuses that highlight the common errors, and analysis of these examples. The first example had an error of 0.157 with our bag of words model. The second example had an error of 0.185 with our GloVe model.



	Bag of Words					GloVe				
	How this 29-year-old makes a living as a professional bridesmaid.					Drama: Joy keeps changing her story...				
	❤️	😆	😮	😢	😠	❤️	😆	😮	😢	😠
Predicted	65	0	18	7	9	72	10	13	5	0
Real	8	67	23	3	0	0	67	3	3	28

The first error occurred because the TF-IDF weighting is going to place a very high value on the word "bridesmaid," without understanding the context in which it's used. This caused a prediction with lots of heart reacts, but in reality more people found it funny or shocking.

The second error occurred because of a similar reason. GloVe normally does well because it uses a massive corpus, but in this case it confused a name for a word. This politically charged post about Joy Reid caused many angry and funny reacts, but "joy" lead the predictor to output many heart reacts.

Conclusion

We developed an approach using an MLP in conjunction with a number of featurizers to predicting realistic proportions of the Facebook reactions for a given Facebook text-based post. We saw that unigram-bigram bag of words performed the best among the featurizers we tested (Word2Vec, FastText, GloVe). Overall, the output from our model (see Qualitative Analysis) produces reaction outputs that look heavily within the realm of what real results. This realistic prediction ability can allow users to filter posts based on their mood or temporal preferences. As well, it allows publishers to get a solid representation of how their posts will be received by their audiences.

Moving forward, we anticipate a number of experiments to attempt to produce even more realistic results. An encoder-decoder-LSTM approach (Hassan et al.) (see Proposed Methods) has been shown to be effective for similar sentiment analysis tasks. Our current database of posts is relatively small, so expanding this dataset would likely improve the Word2Vec and FastText approaches. As well, packaging the approach for usage on a single entity's (i. e. single news source) predictions – to take into account the bias of audiences for the given entity – could allow publishers to utilize this system even further now.

References

[1] Facebook Graph API. 2018, May, from https://graph.facebook.com/

[2] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou, 2018, Achieving human parity on automatic chinese to english news translation. CoRR abs/1803.05567.

[3] J. Akaichi, Z. Dhouioui, and M.J. Lopez-Huertas Perez, "Text mining facebook status updates for sentiment classification," in System Theory, Control and Computing (ICSTCC), 2013 17th International Conference, Sinaia, 2013, pp. 640-645.

[4] Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," , JMLR 12, pp. 2825-2830, 2011.

[5] Pennington, Jeffrey, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)* 12

[6] Woolf, M. (2016, June 26). Classifying the Emotions of Facebook Posts Using Reactions Data. Retrieved June 11, 2018, from http://minimaxir.com/2016/06/interactive-reactions/