CS230

Deep News: Scoring Articles by Quality

Harper Carroll and Susannah Meyer
Department of Computer Science
Stanford University
CS230 Spring 2018
harper@cs.stanford.edu | smeyer7@stanford.edu

Abstract

Finding quality journalism online is becoming increasingly more challenging to navigate. In the age of digital consumerism, consumers are fed countless news stories but have no baseline for evaluating which articles are worthy of their attention. We investigated how to assign news articles scores based on their quality in order to identify articles of high value that contribute more uniquely to a consumer's news experience. We built an RNN model with a single-layer LSTM unit to assign scores between 0 and 1 to a dataset of news articles. We found that such a model does a decent job of scoring articles given a similarity threshold of 0.1. However, there is room for improvement likely due to a number of constraints relating to data, labels, and model complexity.

1 Introduction

The news feeds of the public are consistently bombarded with shallow, copy-and-pasted news articles with the intention of driving up traffic for publishers' revenue; relatively few news sources genuinely produce new value in the online ecosystem. The rise of digital journalism is transforming the news industry into one that values quantity over quality, and consumers continue to face challenges in sourcing articles of consistently high value.

This leads to a number of problems within the news industry, including a mismatch of advertisement quality and article quality, loss of time and efficiency for the public, and the spread of low-quality articles that has begun to define the entire sphere of journalism. Such low-quality articles consist of commodity news, in which publishers imitate news content from online with no journalistic effort involved in order to produce a certain output volume of news, as well as articles that might be classified as "fake news." Fake news has become an especially grave issue around the world, having far-reaching political, social, and environmental effects. Our project aims to tackle these issues; we set out to create an Automated Article Scoring (AAS) system to automatically assign quality scores to articles with the hope of providing a metric for quickly evaluating articles based on the quality of their text.

The input to our model is the main text of an article, which has been extracted from the article's URL and transformed into a list of word IDs. We then use an RNN model with a single-layer Long Short-Term Memory (LSTM) unit to output a predicted article quality score between 0 and 1.

2 Related work

To prepare to approach this problem, we investigated previous related works to consider how previous scholars approached similar scoring problems and the results their models produced. Three Google

CS230: Deep Learning, Spring 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

engineers found that compared to the use of a deep neural network, sequence to sequence learning was enhanced with the use of a multilayered LSTM to map an input sequence to a vector of a fixed dimension, and another deep LSTM to decode the target sequence from the vector [1]. To solve a problem similar to ours, a group of engineers from Cambridge created a neural network for Automated Text Scoring (ATS). Alikaniotis, Yannakoudakis, and Rei (2016) created a model that forms word embeddings by learning the extent to which specific words contribute to the text's score and weighting words appropriately [2]. This group also used LSTM networks to encode text meaning, and they achieved excellent results compared to other cutting-edge techniques. Two engineers at the University of Singapore solved a similar task, Automated Essay Scoring (AES) [3]. Compared to other models such as convolutional neural networks (CNNS) and RNNs using gated recurrent units (GRUs), their model based on LSTM networks performed highest, outperforming the baseline significantly without any manual feature engineering.

These two tasks of essay and text scoring address the same foundational technical problem as the problem of article scoring at hand - that is, assigning a concrete score to a block of plain text. However, these use cases involve differently framed applications and give weight to different parts of text; for example, AES requires giving chosen weights to various grammatical structures. In our research, we found that there is seemingly no work within the literature of assigning scores to text of specifically journalistic endeavors; however, work has been done with similar goals of efficiently distributing news content in mind. Zhang, Yao, and Sun completed a survey of deep learning techniques used to implement Content Recommendation Networks (CRNs), which aim to distribute content to consumers based on a number of preferences; the group found that a model known as the Recurrent Recommender Network (RRN) was successful in building on top of an RNN model with two LSTM networks to model the seasonal evolutions of items and changes of user preferences over time [4]. Similarly, our project aims to predict quality scores for articles in order to standardize a metric to help consumers interact solely with news of a certain quality.

3 Dataset and Features

We began with 55,000 examples in the form of article URLs from the Dallas Morning News publisher. Our dataset was courtesy of JSK Fellow, Frederic Filloux, and each article within the dataset was provided with an associated quality score that comprises the ground-truth score we have used to train our model. This ground-truth score, labeled as the Overall Score in our dataset, is a weighted average calculated by Mather Economics averaging four other determined scores. These four scores include the Reach Score, Quality Score, Core Audience Score, and Yield Score, descriptions of which can be found outlined in Figure 1 below. Each of these scores define a number of user engagement metrics, including the scroll-depth of a user's interaction with an article, the number of page views an article received, and the amount of revenue generated by the article. We then used the Overall Score, a weighted average of these four scores recording such metrics, as a proxy for the quality score of an article. The Overall Score labels originally spanned a range from 0 to 100 with a few outliers reaching into the thousands. To process these scores, we normalized the Overall Score of each articles to fit into a range between 0 and 1, inclusive.

A great deal of preprocessing was required before being able to feed inputs into our neural network. Using the Python Goose Extractor, we extracted the main textual content from each given URL. As the Goose Extractor was unable to parse articles that relied fully on multimedia, this preprocessing step left us with a number of empty article text files; because of this, it was necessary to write a script to delete all empty article text files from our entire dataset in order to avoid scoring blank articles. As a result, we ended up with a total of 50,300 non-empty article text files. We then used a tokenizer from Python's Natural Language Toolkit (NLTK) to transform the string of main text for each article into word tokens. Next, we created a word-to-ID dictionary to map every word token to a unique ID, including one <PAD> word that was needed later on to pad each training example to be of equal length. We then used pretrained GloVE word embeddings to create a dictionary that mapped every word ID to a 100-dimensional embedding vector. Therefore, the features of each article were the list of word IDs as 100-dimensional embedding vectors, which were extracted using a Tensorflow embedding lookup with arguments being the aforementioned embedding dictionary and article word ID lists.

We split our training/validation/test sets into a 70/20/10 ratio.

Column	Description
URL	URL
Overall Score	Score averaging all scores below
Reach Score	Volume of: page views, non-direct and non-referrer page views
Quality Score	Average scroll depth, average time per page
Core Audience Score	Proportion and volume of: known and local page views, direct and internal first referrer page views, page views from users in the top 2 engagement buckets
Yield Score	Volume of: ad revenue, conversions from an article, pages on the path to conversion

4 Methods

To build our model, we implemented a dynamic Tensorflow RNN model with a single LSTM layer with 100 hidden units using a tanh activation function; the outputs of our LSTM layer were then fed into a fully connected layer with a sigmoid activation function. An RNN, a recurrent neural network, is a network that can take advantage of its internal state in order to process sequences of inputs; an LSTM unit, a long short-term memory unit, is a kind of building block for RNN layers that is capable of learning long-term dependencies.

To feed in our training examples as inputs to our model, we needed to pad each example so that each article within a training batch was comprised of the same number of word tokens. To do so, we added '<PAD>' as a word into our vocabulary in our preprocessing step of creating word-to-ID dictionaries and embedding vectors. Each article with a number of word tokens below the maximum article length, then, had a number of <PAD> tokens concatenated at the end of the article to standardize input length. After using an embedding lookup to map each article's list of word tokens as IDs to a corresponding 100-dimensional GloVe embedding vector, we passed our inputs into an LSTM layer. The outputs of this layer were then masked in order to ensure that values of the LSTM's output corresponding to <PAD> tokens were not accounted for in calculating values for our cost function. Finally, our model's fully connected layer uses a sigmoid activation function to output score predictions.

To train our model, we calculated a mean squared error cost function:

Mean squared error = $\frac{1}{n}\sum_{i=1}^{n}(prediction_t - score_t)^2$

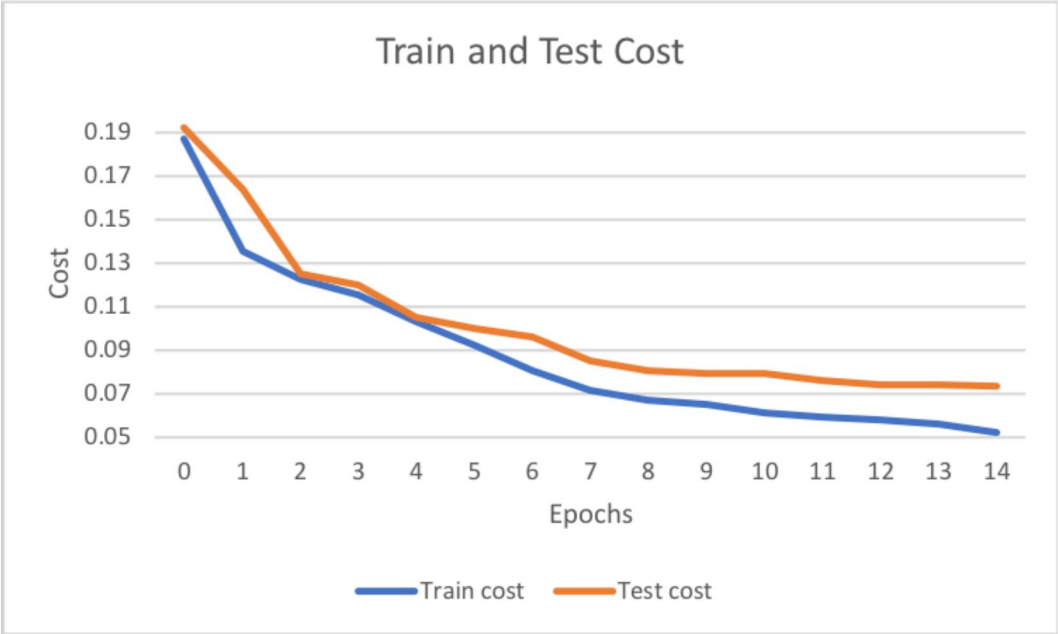
This cost function measures the average of the squares of our model's errors (i.e. the difference between the squares of our model's predictions and the squares of the examples' true scores). To minimize cost, we used an Adam optimizer.

5 Experiments/Results/Discussion

In order to optimize the performance of our model, it was necessary to tune a number of hyperparameters. The hyperparameters we tuned methodically include the number of LSTM hidden units, learning rate, and batch size. In order to select final values for these hyperparameters, we simultaneously ran a series of models on Farmshare, Stanford's community computing environment. Each model ran against a control and had a single variation in the value of one hyperparameter according to a range of values that we selected beforehand according to appropriate scales (i.e. selecting from a logarithmic scale when tuning learning rate, etc.). Hyperparameter values that led to higher performance on our

validation set were then selected for our final model running with the test set. The values that led to higher performance were 100 LSTM hidden units, a learning rate of 0.0001, and a batch size of 100.

To measure performance, we defined a performance metric dependent on a similarity threshold. Because we capped our ground-truth labels to be within a range of 0 and 1, we decided on a similarity threshold of 0.1 to determine correctly scored articles. That is, a single article in our model was classified as correctly scored if the model's prediction diverged from the article's ground-truth score by a maximum of 0.1. We decided on this threshold to be strict enough to hold confidence in the training of our model while lenient enough to avoid an unreachable level of precision. Below are the final results we found in terms of cost and performance.



Our results show that our model outputs high performance rates on the training set and outputs lower performance rates on the test set. In testing a number of different variations on our model, we found that having an LSTM layer as opposed to a basic RNN layer improved performance; likely, the capability of the LSTM to deal with long-term dependency helped our model. Regardless, these results leave room for improvement in terms of test set performance, which suggests a potential problem of overfitting to our training set. Additional layers of complexity and the implementation of regularization techniques like dropout regularization might help mitigate this for the future.

We have drawn a number of interpretations from these results. Regardless of the issue of an overfit, the extraction of main text from the URLs of each article in our dataset relied on the Python Goose Extractor library to parse through the HTML of each article and keep only plain text within the article's body. The extractor was unable to parse articles which fully relied on multimedia, and it was impossible for us to manually sanity check the parsing of those articles for which extraction succeeded. This might lead to unstable results.

Next and importantly, the dataset of our model provided the aforementioned Mather Economics scores related to user engagement as our ground-truth quality labels. Our results might show that extracting features using only an article's text from a natural language processing standpoint are inadequate for predicting such scores. For example, consider the situation of an article going viral. Oftentimes, the reason for virality is influenced by celebrities, politicians, or other publicity factors completely outside of relation to the actual article text itself. If an article in our training set or test set demonstrated a particularly high score due to a similar external factor, there would be no feature within the article text to indicate that. The fact that our model could not fully predict accurate scores given article text, then, perhaps suggests that user engagement and actual article content should be evaluated separately. This leads to further questions surrounding the state of journalism and questioning the factors that lead to article popularity outside of quality and textual effort.

6 Conclusion/Future Work

Our final model taking advantage of an LSTM layer with 100 hidden units demonstrated the highest performance, and our results suggest that there is more work to be done in terms of expanding and iterating on our model as well as considering alternate routes in data labeling that do not rely on user engagement metrics.

With more time and resources to continue this project in the future, a first step we would take is to investigate human-labeled quality scores for better ground-truth labeling that directly relates to text quality. Our industry mentor, Frederic Filloux, has begun work on a human-scoring interface which would make this possible in the near future. A crowdsourced dataset of articles across many publishers with experts in the industry robustly scoring each article on text quality would better refine our task and lead to more narrowed interpretations of results.

With these improved quality scores, we would propose a number integration into applications that would benefit the industry as a whole. First, smart advertising could effectively match the revenue of advertisements with the quality of an article, better incentivizing publishers to produce content of a higher level. Further, this project could integrate into a personalized news experience based on level of article quality, as well as into the enhancement of smarter news aggregators to remove low quality sources. Generally, a universal quality indicator for news distributed on the web would save consumers accumulated hours of time and legitimize high quality news sources.

7 Contributions

Harper Carroll: I conducted research on past works to develop a baseline model for our own problem. I developed and debugged our initial RNN model using Tensorflow. I wrote a pre-preprocessing script, run before running our main model to randomly divide all 55,000 articles (with their associated 55,000 article label files) into 70/20/10 train/dev/test sets, accessed easily by the model from their respective folders. I worked to iterate hyperparameter tuning.

Susannah Meyer: I developed our initial RNN model using Tensorflow. I also created a job on Farmshare to tokenize our entire dataset of articles in order to avoid downloading the set of 55,000 articles locally and in order to extract the article's main text in tokenized form for model input. I

also worked to delete any empty files that were left as a result of failed text extraction and worked to iterate hyperparameter tuning.

References

[1] Sutskever, I. & Vinyals, O. & Le, Q.V. *Sequence to Sequence Learning with Neural Networks.* Google.

[2] Alikaniotis, D. & Yannakoudakis, H. & Rei, R. (2016) *Automatic Text Scoring Using Neural Networks.* Cambridge, UK: University of Cambridge.

[3] Taghipour, K. & Ng, H.T. (2016) *A Neural Approach to Automated Essay Scoring.* Austin, Texas: Association for Computational Linguistics.

[4] Zhang, S. & Yao, L. & Sun, A. (2017) *Deep Learning based Recommender System: A Survey and New Perspectives.* ACM J. Comput. Cult. Herit. 1, 1, Article 35.

[5] Pennington, J. & Socher, R. & Manning, C.D. (2014) *GloVe: Global Vectors for Word Representation.*

[6] Bird, S. & Loper, E. & Klein, E. (2009) *Natural Language Processing with Python.* O'Reilly Media, Inc.

[7] Goldberg, Y. & Levy, O. (2014) *word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method.*

[8] Abadi, et al. (2015) *TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.* tensorflow.org.