CS229 Final Report

Application of Sentiment Analysis to Labeling Characters as Good or Evil

Ethan Curtis, Maria Karelina, Felipe Kettlun

Abstract

Sentiment analysis models were trained with the objective of transferring this learning onto determining if characters in books are good or evil. Typically, sentiment analysis aims to test on a set-apart subset of the training set, however, with a small, hand-labeled training set, this approach was infeasible. Instead, we employed common models to learn sentiment on the training database, then developed code to label the sentiment of characters in books based on the sentiment of sentences in which they appeared. We screened models and datasets to determine which model/training dataset combination generalized most accurately to the other sets, then applied the resulting models to characters in books. In spite of trying several different models and character-labeling methods, none produced accuracies above 0.50.

Introduction

Have you ever found yourself wondering whether Sauron or Pippin was the real villain of Lord of the Rings? As children, we were taught that everything could be divided into two categories: good and evil. We intend to use sentiment analysis to analyze characters from children's books on how good or evil they are.

Generally, when designing a supervised classifier it is best to train the model on the same type of data as would be used for testing and predictions. However, there are no available datasets that have characters from books labeled as good or evil. This means that we have a very limited dataset of labeled characters. So instead we focus on creating a model that can be trained with available labeled datasets and transfer the model onto classifying characters.

It is important to note that a key of this project is to not simply create the best sentiment analysis for a specific dataset, but to train a model in a way that could give us more transferability. We must use a sentiment database such as Amazon Book reviews, Twitter, or IMDB reviews to train a model to predict binary sentiment (0,1). Then we must find a simple way to attribute text to a character, take these texts, label them with a sentiment using our previous model and average the score to rate characters as good or evil.

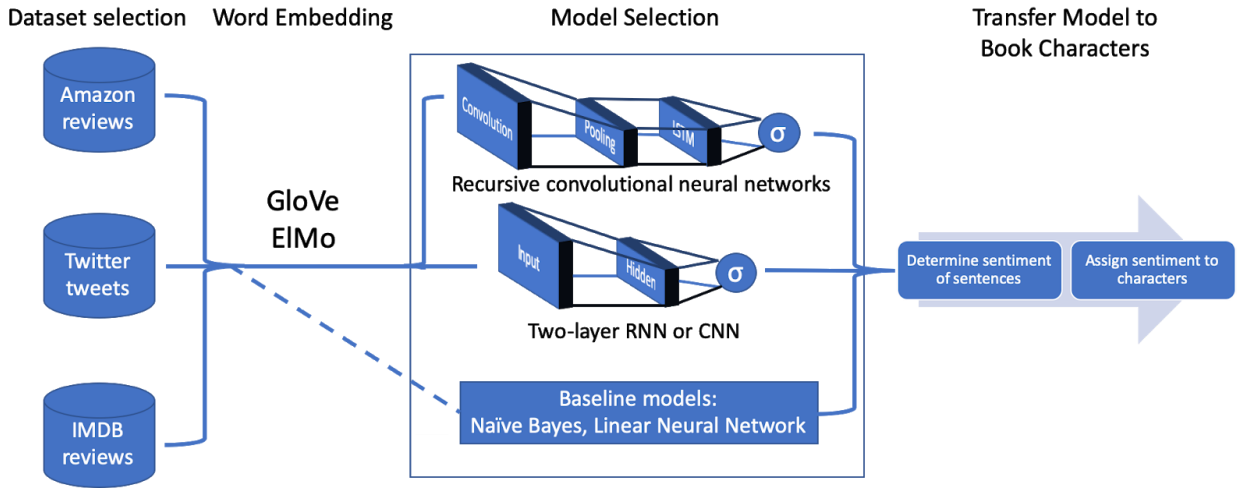

Figure 1. General pipeline to describe the process of our project. We first pick a dataset to train with that produces the most general models. We then compare how different embeddings perform on sentiment analysis tasks. Then we compare different classification models and their transferability. Lastly we split books into sentences that we assign to characters and see how our model performs.

Related Work

Natural language processing has focused on the problem of sentiment analysis, leading to a plethora of literature using every imaginable featurization and classification algorithm [1]. More classical approaches, such as Naive Bayes [2] (NB) and Support Vector Machines [3] (SVM) using bag-of-words (and similar featurizations like TF and TFIDF) predominated until the mid-2010's. These approaches have the dual advantages of simplicity and interpretability: the underlying mathematical models are straightforward (and therefore easy to implement), and the features and weights of

the models can be interpreted as ranking the sentiment impact of words. However, the underlying assumptions of these models are crude: bag-of-words neglects word order, so NB and SVM cannot capture context. These limitations can be somewhat alleviated by including ngrams (word groups of length n) as features, but the accuracy improvement is marginal [4]. Increasing the number of features results in minimal training data for each feature (a phrase is less common than a word, which itself may only appear a few times in the training set), which can cause models to fail to generalize beyond the train set.

Deep neural networks using word-vector embeddings have been popularized recently and can provide a significant improvement in accuracy over NB and SVM [5]. These networks can include context in several ways: through the use of convolutional layers, through context-dependent embeddings, and through the addition of recursion. The input into a convolutional layer is a matrix: one column for each word. Convolutional layers convolve across the columns so that the output of each node is dependent on every word. A typical convolutional neural network (CNN) employed with the GloVe word-vector embedding can improve accuracy by 20% over bag-of-words SVM, and by 5% over GloVe SVM on Twitter datasets [6]. This statistic reveals how CNN outperforms SVM, but also the effect of word-vector embeddings. These embeddings will be discussed in further detail in the next section. Embeddings such as ElMo use a recursive model to embed words, resulting in context-dependent vectors, in contrast to vector dictionary embeddings like word2vec. This difference can provide accuracy improvements of up to 5% [7]. Most recently, marginal improvements to accuracy have been obtained by adding a recursive layer to a CNN. This recursive layer can be added either before the convolution (RCNN) or after it (CRNN) [8], [9]. The ability of nodes to remember previous inputs explicitly incorporates context, increasing accuracy. State-of-the-art models use some combination of these methods and features to incorporate context, as well as additional layers, featurizations, and data processing tricks; there are too many to enumerate here (see [5] for an overview of some such methods).

Dataset and Features

Datasets:

The training set for the sentiment analysis models was comprised of three publicly-available sentiment databases. **Amazon Books** [10]: This dataset contains 970k book reviews taken from Amazon.com. We deem a positive review to be a rating of 4 or 5 stars and a negative review 1 or 2 stars. Neutral reviews are not included in the dataset. There is a bias for positive reviews in the data set, where over 80% of the reviews are positive.
IMDb [11]: This dataset contains movie reviews along with their associated binary sentiment polarity labels. It contains 50k reviews split evenly into 25k positive reviews and 25k negative reviews. No more than 30 reviews are allowed for any given movie because reviews for the same movie tend to have correlated ratings. IMDb reviews have scores 1-10. Scores <= 4 are considered negative and >=7 are considered positive. Neutral ratings are not included in the dataset.
Twitter [12]: This dataset contains 1.6 million tweets with emoticons removed along with trinary sentiment labels (positive, negative, neutral). For the purposes of this study, neutral tweets are omitted. The labels were automatically generated using (among other things) the positive/negative nature of the emoticons which were subsequently stripped.

These datasets had commas and periods removed, were converted to lowercase, and split by spaces. Exclamation marks and question marks were separated to be their own token. No treatment of misspelled words, non-ASCII characters, or slang was performed. Such tokens do not appear in the books which form the test set, and therefore should not affect the results. For the Amazon dataset, it was necessary to remove some HTML tags. Test sets were formed from each dataset by removing ~400 examples.

For testing the character labeling, we created our own small dataset consisting of 6 books and 20 characters. We took the text of books of the public domain and manually labeled major characters in the book as good (1) or evil (0). It is also important to note that there are several names that a character can be referred to as, so our labels consisted of a score: 0 or 1 and a list of names that belong to that one character. The books we used are: “Ashputtel,” "The Phantom Tollbooth," "Beauty and the Beast," "The Giving Tree," "The Count of Monte Cristo," and "Hamlet."

Features and Embeddings:

The Naive Bayes, logistic regression, and simple neural network all used the bag-of-words featurization of sentences. A dictionary of all words was created, and each sentence was converted to a vector indicating which words were in the sentence. Two different embeddings, GloVe [13] and ELMo [14], were used to embed words as vectors and convert sentences into matrices. Static embeddings like GloVe and word2vec are designed to incorporate word similarity by mapping words to vectors based on word co-occurence (the dot product of two words in GloVe is the log probability of co-occurence). Context-dependent embeddings, like ELMo, rely upon a deep, recursive, pretrained model. Sentences are

fed through this model, which embeds words based on the internal states of the recursive neural network, leading to embedding values which are context-dependent. Additionally, ELMo is character-based, which allows for it to capture the meaning of out-of-dictionary tokens.

Methods

Sentiment Classification:

For the sentiment classification we implemented four algorithms:

1) **Naive Bayes:** This model was chosen as a baseline because of its simplicity and low computational requirements. It assumes that the input data is conditionally independent given the labels. We also used Laplace smoothing and only words that appeared at least five times across the dataset were considered.
2) **Linear Neural Network (LNN):** A linear neural network bridges the gap between logistic regression and deep neural networks. After a few iterations we ended up with a model consisting of five layers. The first layer has 1,000 neurons and the following layers all have 2,000 neurons. We used ReLu activation functions for all layers. For the output we used a sigmoid function. To control overfitting we used L2 regularization and dropout.
3) **Convolutional Neural Network (CNN)**: A specific architecture of neural networks in which we use filters to avoid the overfitting problem of fully connected layers. This allows for the model to generalize learning. For example, if there is a phrase with a given meaning, it probably does not matter if the phrase is at the beginning of the text, in the middle or at the end. The filter is used to perform a sliding dot product with the input data. In our implementation we used two filters of size 64. We also used a max pooling layer of size 4, which takes neighborhoods of 4 entries and outputs the maximum value among them.
4) **Convolutional Recurrent Neural Network (CRNN)**: This model is a CNN with a long short-time memory (LSTM) layer of size 70. A LSTM implements cell units that can remember data for arbitrary intervals of time. Forget and update gates are used to control the value stored in the memory cell. Because of this memory cell LSTM do not suffer from vanishing gradient as traditional RNN.

LNN, CNN and CRNN were implemented using Keras python framework.

Character Sentiment Assignment:

In order to rate characters we first assign sentences or sections of text to the character, then evaluate the chunk of text with our sentiment models. We averaged all the sentences associated with a character and used this number as the moral rating of the character. We use two different ways of assigning text to characters we will call these split 1 and split 2. For split 1 we split the book by periods and assign the sentences to all the characters whose names appear in the sentence. Note that there is other punctuation other than periods in the texts, however, we only use periods to generate sentences that are slightly longer and also because sometimes exclamation marks and questions tended to be short and related to the sentences before or after. For split 2 we assign sentences to characters by finding a character's name and creating a sentence by taking 10 words before and after.Note that in one method one sentence may be the same for several characters in the other all sentences will be somewhat different between characters.

Experiments/Results/Discussion

Testing Sentiment Analysis Models :

We tested and trained several different models on the same dataset as a first-pass comparison between methods, and to gain a baseline for expected performance on the datasets (Table 1).

	Dataset		
Model:	**Amazon**	**IMDb**	**Twitter**
Naive Bayes	0.907	0.855	0.791
Linear NN	0.896	0.893	0.795
CRNN	0.807	0.880	0.955
CNN	0.888	0.791	0.945

Table 1. Test to determine accuracy of different sentiment analysis methods.

Data selection test:

We wanted to select a dataset with a high signal (sentiment information) to noise (dataset-specific information) ratio so that models trained on that dataset would be transferable. A convolutional LSTM neural net was trained on each dataset. Then, each trained model was tested on a test set from each dataset (Table 2).

From this experiment, we chose IMDB as the most transferable training dataset. Although our CRNN model trained with twitter had very high accuracy when predicting twitter data, it performed poorly when tested against other datasets. Indicating that the model was learning something that is inherent to tweets and not to text found in other contexts. Training on IMDB appeared to be most generalizable, predicting both Amazon and Twitter fairly well (or, at least, better than any other dataset).

	Model trained with:		
Model tested against:	**IMDB**	**Amazon**	**Twitter**
IMDB	0.807	0.673	0.499
Amazon	0.835	0.880	0.115
Twitter	0.563	0.117	0.955

Table 2. Training and testing across datasets on a CRNN model. We can see that training with IMDB produces the most accurate predictions of other datasets.

Embedding Tests:

We also tested embeddings could affect prediction accuracy (Table 3). We used the IMDB database and trained it with both Glove (200 dimension) and ELMo (256 dimension, outputs of the last two layers were flattened to form a 512 dimension vector for each word) embeddings. While ELMo does outperform GloVe, the modest improvement could not justify the severe computational cost.

	CNN	**CRNN**
GloVe	0.675	0.682
ELMo	0.640	0.778

Table 3. Comparison of accuracy with different embeddings when training on CRNN with IMDB data.

	Dataset tested on:		
Model trained	**IMDb**	**Twitter**	**Amazon**
Naive Bayes	0.842	0.586	0.801
Linear NN	0.893	0.580	0.759
CRNN + GloVe	0.807	0.560	0.835
CNN + GloVe	0.791	0.490	0.823

Table 4. Comparing transferability of different models all trained with the IMDB training dataset. We see that transferability is fairly similar between the different models.

Model selection tests:

Once a training database and embedding had been chosen, we decided to also compare how transferability was affected by different models. We trained each model on IMDB training data, then tested on every dataset (Table 4).

Character predictions:

The last test was to determine accuracy of our models when they were used to analyze sentences belonging to a character (Table 5). See methods for details of each splitting algorithm.

	Sentence to character assignment:	
Model	**Books Split 1**	**Books Split 2**
Naive Bayes	0.45	0.45
CRNN + Glove	0.40	0.50

Table 5. Testing character sentiment predictions and comparing between splitting 1 and split 2.

Our models performed poorly when tested on book characters regardless of how we assigned sentences to characters. Here is an example from Ashputtel (also known as Cinderella in other versions of this story) where we see some potential failures for this model:

"*Shake, shake, hazel-tree, gold and silver over me! then her friend the bird flew out of the tree, and brought a gold and silver dress for her, and slippers of spangled silk; and she put them on, and followed her sisters to the feast.*"

The sentiment of our CRNN model predicted this sentence to be 0.86080. However the only character name that appears in this sentence is that of the "sisters" and not "Ashputtel" which the "her" refers to. The subject of a sentence hard to identify partially because pronouns are hard to track when texts are split into sentences without context. Additionally when multiple characters are present in a sentence it is hard to identify who the sentiment of the sentence is referring to.

The relationship between sentiment of a sentence and the moral value of the associated character is not always clear. For example, almost every model rates Ashputtel lower than her sisters, because Ashputtel spends much of the story wearing rags and struggling, associating her with sentences with negative sentiment. For example:

"In the evening, when she was tired, she had no bed to like down on, but was made to lie by the hearth among the ashes, and as this of course made her always dusty and dirty, they called her Ashputtel."

Conclusions and Future Work

We found that we can create fairly accurate sentiment analysis with simple models but these models do not perform well when transferred to other sets of analogous data. The lack of labeled character data is a major bottleneck for building a character classifier, and it is not easy to transfer sentiment to determine character morality.

We could try using Natural Language Toolkit, NLTK, to create syntax trees to identify subject in a sentence in order to more accurately assign sentiment [15]. Once we have a more accurate sentiment transferring technique, then we can begin to optimize the structure and parameters of our neural networks. We did not see significant improvement over classical methods because the neural networks were not fully optimized for this problem.

We could also consider using a semi-supervised learning technique with our small amount of labeled data. We could do this by clustering characters, then using a few known labels to determine the rest of the labels.

Contributions

https://github.com/maschka/cs229/

Ethan Curtis - Cleaning of Twitter dataset. Implementation of GloVe and ElMo embeddings, CNN and CRNN.
Maria Karelina - Cleaning of Amazon dataset. Implementation of Character Sentiment Assignment methods.
Felipe Kettlun - Cleaning of IMDb dataset. Implementation of Naive Bayes and LNN.

References

1. M. Mäntylä, D. Graziotin, M. Kuutila. “The evolution of sentiment analysis—A review of research topics, venues, and top cited papers.” *Computer Science Review*, vol. 27, 2018, pg. 16-32.
2. J. Singh, G. Singh, R. Singh, “Optimization of sentiment analysis using machine learning classifiers.” *Hum. Cent. Comput. Inf. Sci.,* vol. 7, no. 32, 2017.
3. A. Hamoud, A. Alwehaib,K. Roy, M. Bikdash. “Classifying Political Tweets Using Naïve Bayes and Support Vector Machines.” In: *Lecture Notes in Computer Science. Recent Trends and Future Technology in Applied Intelligence*, vol. 10868, M. Mouhoub, S. Sadaoui, O. Ait Mohamed, M. Ali, Eds. Springer, Cham, 2018.
4. A. Tripathy, A Agrawal, S. Rath, “Classification of sentiment reviews using n-gram machine learning approach.” *Expert Systems with Applications.,* vol. 57, no. 15, 2016.
5. L. Zhang, S. Wang, B. Liu, “Deep learning for sentiment analysis: A survey.” *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, 2018.
6. Z. Jianqiang, G. Xiaolin, Z. Xuejun, “Deep Convolution Neural Networks for Twitter Sentiment Analysis.” *IEEE Access*, vol. 6, 2018.
7. F. Nurifan, R. Sarno, K. Rossa Sungkono, “Aspect Based Analysis for Restaurant Reviews Using Hybrid ELMo-Wikipedia and Hybrid Expanded Opinion Lexicon-SentiCircle.” *Int. Jour. Intelligent Eng. & Sys,* vol. 12, no. 6, 2019.
8. N. Chen, P. Wang, “Advanced Combined LSTM-CNN Model for Twitter Sentiment Analysis.” In: *5th IEEE International Conference on Cloud Computing and Intelligence Systems, Nanjing, China, Nov. 23-25 2018.*
9. Z. Zhang, D. Robinson, J. Tepper, “Detecting Hate Speech on Twitter Using a Convolution-GRU Based Deep Neural Network.” In: *Lecture Notes in Computer Science. The Semantic Web ESWC 2018,* vol. 10843. A. Gangemi et al., Eds. Springer, Cham, 2018.
10. J. Blitzer, M. Dredze, F. Pereira, “Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification.” In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Prague, Czech Republic, June 2007.*
11. A. Maas, R. Daly, P. Pham, D. Huang, A. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis.” In: *The 49th Annual Meeting of the Association for Computational Linguistics, 2011.*
12. A. Go, R. Bhayani, L. Huang, “Twitter Sentiment Classification using Distant Learning.” *Stanford CS224N,* 2009.
13. J. Pennington, R. Socher, and C. Manning, “GloVe: Global Vectors for Word Representation.” In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, October 2014*.
14. M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, “Deep contextualized word representations.” *Proceedings of NAAC*, *2018*.
15. S. Bird, E. Loper, E. Klein, “Natural Language Processing with Python.” O’Reilly Media Inc, 2009.