

Classifying the Political Lean of Reddit Posts

Varun Singh, Kevin Lam, Sreetama Chowdhury

Code:

https://drive.google.com/drive/folders/1myuv89cdpbUHqosJy4liqLZbw57LWPIS?usp=share_link

Demo:

https://drive.google.com/drive/folders/1AR-vptsBe995SeYfTnKfME-psbUEBwL5?usp=share_link

Introduction

The task this project is attempting to carry out is the identification of the political lean of a given social media post via its textual content. For simplicity's sake, it is a binary classification task — classifying a post by political alignment, whether the content of the post leans more towards liberal or conservative ideas. More specifically for our project, inputting the title of a Reddit post (distinct from the body of the post, but more reliable because not every post on Reddit has a body at all) should provide insight into its political alignment. We found the categories of “liberal” and “conservative” to be useful because they are broader and less region-specific than the standard American categories of “Democrat” and “Republican.”

We trained three models to classify these posts — a baseline model, which was logistic regression with bag of words, a recurrent neural network, and a convolutional neural network — and compared their results.

Models

This section was written with the assistance of ChatGPT.

Our baseline model for text classification was logistic regression with a bag of words approach for generating embeddings. This model is a simple and effective way to classify text by using a linear model to predict the probability of a given text belonging to a particular class.

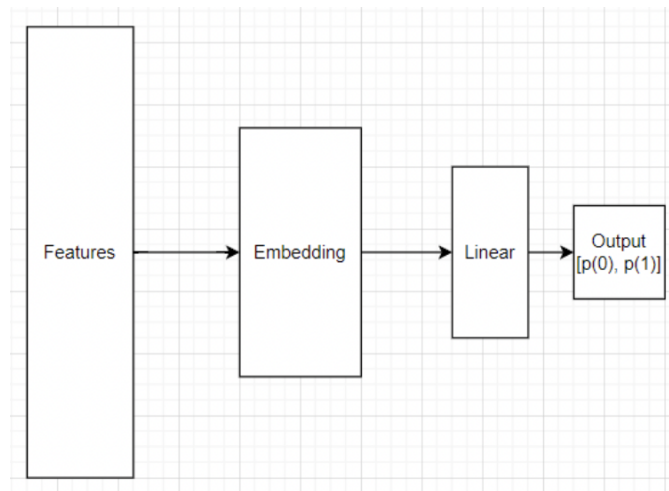
The first step in this model is to create an embedding layer, which is used to derive the semantic meaning of the words in the text. This is accomplished by using the bag of words algorithm, which captures the frequency of words in the vocabulary in the given sentence.

Once the sentences have been embedded, a single fully-connected linear layer is applied to the embedded vectors. This layer, which is typically represented by the equation $aw + b$, produces a

vector of probabilities for each class. In this case, the output vector will have two values, $[p(0), p(1)]$, representing the probabilities that the text belongs to the first or second class, respectively.

One important aspect of this model is that it does not use a softmax layer to normalize the probabilities. This is because the model is using a cross-entropy loss function, which is capable of directly maximizing the likelihood of the correct class. This allows the model to be trained more efficiently, and it also ensures that the probabilities produced by the model are well-calibrated.

A diagram of the baseline model can be seen below:



Our second model for the text classification was a recurrent neural network (RNN). This model is a more complex approach that uses sequential information to better classify text.

The first step in this model is the same as the baseline model, where an embedding layer is used to derive the semantic meaning of the words in the text. This allows the model to capture the relationships between words and use this information to make predictions. We attempted to use this model with the bag of words embedding technique, but we were advised to use embeddings that preserved sequential information as well. Therefore, in addition to this RNN, we made another version of the model. The embeddings we used this time were embeddings taken from Stanford's GloVe project. The embeddings for the words were arranged side-by-side and passed into the model as an embedded sentence.

Once the words have been embedded, the embedded vectors are passed through a hidden layer, which is implemented using PyTorch's RNN module. This layer is the default RNN implementation in PyTorch, the documentation for which can be seen below.

RNN

CLASS `torch.nn.RNN(*args, **kwargs)` [\[SOURCE\]](#)

Applies a multi-layer Elman RNN with `tanh` or `ReLU` non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time t , x_t is the input at time t , and h_{t-1} is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0. If `nonlinearity` is `'relu'`, then `ReLU` is used instead of `tanh`.

The output of the RNN is then passed through a linear layer, which produces a vector of probabilities for each class. In this case, the output vector will have two values, $[p(0), p(1)]$, representing the probabilities that the text belongs to the first or second class, respectively.

Just like the baseline model, this RNN model does not use a softmax layer to normalize the probabilities, since the model is using a cross-entropy loss function.

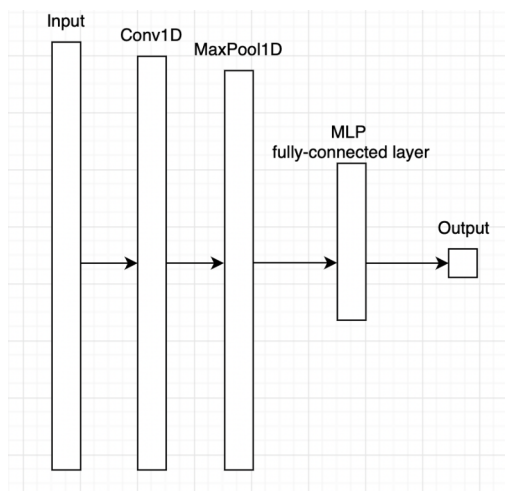
Our third model for this text classification task used convolutional neural networks (CNNs). This approach involves using a "sliding window" of weights, which are learned by the neural network, to process the input data.

The first step in our CNN was to apply a convolutional layer to the input data. This layer uses a set of weights, which are learned by the network, to create a new representation of the data. The weights are applied to the input data using a "sliding window" approach, where the weights are moved across the input in small steps to produce a new representation of the data.

After the convolutional layer, a max-pooling layer is typically used to reduce the dimensionality of the data. This layer uses a "sliding window" to extract the maximum value within a small region of the data. This reduces the size of the data, which allows the network to more efficiently process the data and reduces the number of parameters that need to be learned.

Finally, the processed data is passed through one or more fully-connected layers, known as multi-layer perceptrons (MLPs), to produce a prediction. For our text classification, the final layer of the MLP consists of a single neuron, which produces a value between 0 and 1 representing the probability that the input data belongs to the positive ("Liberal") class.

A diagram of the architecture for the CNN can be seen below:



In our experiments, we found that using CNNs with bag-of-words embeddings and GloVe embeddings produced similar performance. This suggests that the choice of embedding technique did not have as important of an impact as the use of convolutional and max-pooling layers.

Dataset

The dataset we used, linked [here](#), was an existing pre-labeled collection of (approximately 13000) political Reddit posts that we found on Kaggle. The dataset contains both the titles of each Reddit post and the text of the post itself – however, as stated previously, a great many of the posts collected don't seem to contain any actual text beyond the title, making the set of titles the larger one (12324 title values in comparison to 2428 post body values). We found that the structure of Reddit posts means that blank bodies, with the entire informational content of the post contained in its title, are still valid, and therefore we trained our models on the set of titles alone.

In potential future expansions of this project using data from other (ie non-Reddit) social media platforms, we might consider splitting title and body data and training the models on the body of the posts, or using any existing post bodies to corroborate the classification of the title. At present, since the titles of these posts tend to be relatively short, we hoped that the words contained therein would have strong semantic relationships with their neighbors, and that our models would be able to effectively learn these relationships.

The posts are time-stamped 2021 or earlier (meaning that our models will therefore not end up being trained with any information pertaining to political instances, viewpoints, and upheaval from the past year), are pre-sorted into “liberal” and “conservative” categories, and are sourced from a wide variety of different subreddits.

Results

In order to evaluate our models, we calculated the precision and recall of all three models on our test set and compared them to each other. We elected to use precision and recall as metrics instead of accuracy, since the input data was skewed, and contained more liberal posts than conservative posts.

	Baseline	RNN	RNN (GloVe)	CNN (GloVe)
Precision	0.798	0.844	0.999	0.635
Recall	0.804	0.789	0.654	0.988

I fed the previous results to ChatGPT and asked it to analyze them. Its response:

The baseline model has a high recall, meaning that it correctly identifies most of the actual Conservative and Liberal post titles. However, its precision is lower, which means that it also misclassifies a relatively large number of post titles as Conservative or Liberal when they are not.

The RNN model has a higher precision, indicating that it is better at correctly identifying the true Conservative and Liberal post titles. However, its recall is lower, meaning that it misses more of the actual Conservative and Liberal post titles.

The RNN (GloVe) model had a much higher precision of 0.999, but a significantly lower recall of 0.654. This indicates that the model was able to accurately identify nearly all of the post titles, but there were many post titles that it failed to classify correctly.

The CNN model has the lowest precision, which means that it is the least accurate at correctly identifying the true Conservative and Liberal post titles. However, it has the highest recall, which means that it correctly identifies the most number of actual Conservative and Liberal post titles.

Overall, the RNN model appears to be the most effective at classifying the political lean of post titles, due to its high precision. However, depending on the specific requirements of the task at hand, the CNN model may also be a good choice due to its high recall, and it may also be worth considering using a different approach, such as the RNN (GloVe) model if a higher level of precision is desired.

Conclusion

This section was written with the assistance of ChatGPT.

Overall, none of our models showed particularly spectacular results. In fact, all of the models had relatively low precision and recall scores, which suggests that they were not very effective at accurately classifying the reddit posts.

The convolutional neural network showed a particularly large difference between its precision and recall numbers. Its precision score was relatively high, indicating that it was good at identifying conservative posts, but its recall score was very low, indicating that it was not very effective at identifying liberal posts. This suggests that the CNN was overfitting to the conservative posts in the training data, and was not able to generalize well to other types of posts. Given the results of this study, it seems that applying convolutional neural networks to this task may not be a good approach if the model needs to be generalizable.

Ultimately, the RNN (base) was our best model. It had high and consistent precision and recall scores compared to the other models. This suggests that RNNs may be a good choice for this task, and further research should be conducted to determine the best way to apply them to this problem.

References

Jeffrey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation," <https://nlp.stanford.edu/projects/glove/>

Neel Gajare, "Liberals vs Conservatives on Reddit [13000 posts]," <https://www.kaggle.com/datasets/neelgajare/liberals-vs-conservatives-on-reddit-13000-posts>