Bias Detection in Sports Articles Using Structured Data

By Sherman Ng (SUNET ID: sqng629)
Email: sqng629@gmail.com, sqng629@stanford.edu

Introduction

Bias detection in news articles is a very difficult task where much research is currently being done [5]. There have been quite a few different attempts at creating language models using machine learning and deep learning techniques such as linear discriminant analysis [4], recurrent neural networks, and LSTM. This experiment attempts to provide a solution to bias detection not by using a language modelling approach, but rather a methodology focused around using a structured text input feature approach to classify whether or not a particular article is biased. The methodology being tested is centered around first extracting text features, such as the text complexity and the number of words of certain parts of speech using tools such as the Stanford POS tagger, and then passing the structured input into some deep learning network architecture to classify the article.

Related Work

The main motivating factor behind this experiment is that there has already been work done showing the possibility of detecting bias inducing features through extraction and analysis of text and part of speech features [2]. There has not been as much work done implementing an algorithm and architecture to classify articles using only parsed text features such as part of speech features, and because it appears that part of speech and other text features can provide discriminating clues to determining bias presence in news articles, this experiment focuses on exploring the feasibility of developing such a model.

Dataset and features

This experiment specifically focuses on bias detection in sports news articles, and the dataset used is a database from the UCI Machine Learning Repository [1] containing 1000 parsed sports articles where part of speech and other text features such as the frequency of exclamation or question marks in the article, the frequency of pronoun use, and the number of foreign words used in the article are extracted. Each article in the database contains 59 total extracted features, so the full input dataset to the algorithms developed in this experiment can be viewed as a 1000 by 59 matrix. 95% of the dataset is used to train the network, while 5% of the remaining data is used as a validation dataset to measure the effectiveness of the trained network.

Here is an example of how the input data for 1 single input example appears:

Article #	Frequency of plural proper nouns	Frequency of base form verbs	Frequency of genitive markers	Frequency of commas	Frequency of foreign words	Frequency of full stops	Text complexity score	……
16	30	45	20	40	5	20	18	

One of the central challenges of this experiment revolves around the small number of input examples available to train various deep learning networks. Because of the small number of example data points available for training, overfitting is a key issue that had to be addressed using a variety of different regularization and data augmentation techniques discussed later in the paper.

Methods

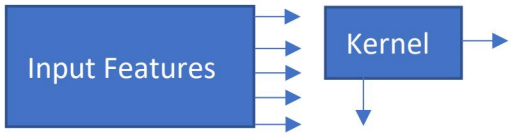
Fully Connected Neural Network

The first model that I experimented with is a simple neural network comprised by 2 fully connected hidden layers, non-linear activation functions at the output of the hidden layers, an output layer, and a sigmoid activation function at the output. An Adam optimizer was chosen for optimization of the weights of the neural network, and Xavier initialization was used for initialization of the weights and zero initialization for the biases. Initially, I used ReLU activation functions in the output of the hidden layers. Eventually, it was determined that using sigmoid

activation functions at the outputs of hidden layers instead produced a better validation accuracy rate. Adding more than 2 hidden layers to the network did not seem to improve validation classification accuracy while increasing the time it takes to train the network, so the number of hidden layers used was kept at 2.

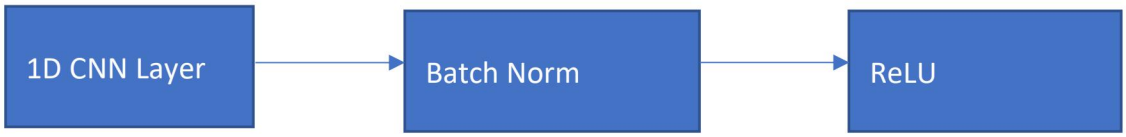
Convolutional Neural Network

Another model that I experimented with in an attempt to create an architecture that potentially generalizes better to unseen examples is a multi-layer convolutional neural network. The 59 input features for each training example are passed as inputs into a network block where the first layer is a 1-dimensional convolutional layer, with kernel sizes and the number of filters determined empirically. In the first convolutional layer, for each training example, c number of m x 1 kernels are convolved with the input features using "same" padding to make sure that input features on the edges do not get dropped out (c and m refer to the number of filters in the convolutional layer and kernel size respectively). The rationale behind using an approach like this is based upon the hypothesis that certain input features may be correlated with each other. Using a CNN approach may enable the model to better capture the interdependencies between the features:



The output of this layer, with dimension p x c x 1 where p is the number of input features, is then passed to a batch normalization layer for regularization purposes.

Each convolutional network block consists of three basic components: a 1-dimensional convolutional layer, a batch normalization layer, and a non-linear activation layer:



Batch normalization should improve performance of a convolutional neural network with multiple layers by reducing the effect of a possible covariate shift in the incoming data. Here is what each batch normalization layer consists of [6]:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The output of the batch normalization step is then fed into a ReLU activation step. The final output of this convolutional network block is then fed into the next stage of the overall network architecture.

Experimentation was done to stack a variable number of convolutional network blocks sequentially, with some network blocks containing a 1-dimensional max pooling layer in between. The output from this chain of

convolutional network blocks is input into multiple fully connected layers with a ReLU activation layer in between them. The output of the fully connected layers is then fed into a final sigmoid activation layer that produces an output prediction of whether or not the input example is classified as a biased article.

Here is how the architecture of the full network appears:

Results

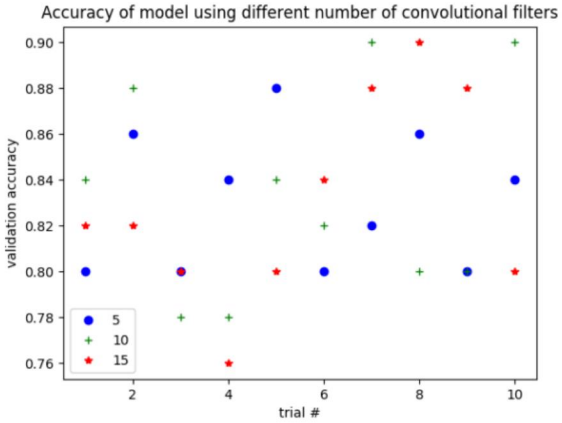
Possibly because of the small number of training examples available for use in training the models, there is quite a bit of fluctuation in the validation dataset accuracy each time a model is trained and evaluated. To get a clearer picture of how a model performs when a certain set of hyperparameters is used, a model is trained 10 times on each set of hyperparameters used, and each time a model is trained, a random 5% of the dataset is selected as the validation set while the remaining 95% is used for training. The performance of the model under a set of hyperparameters is then noted over the 10 trials.

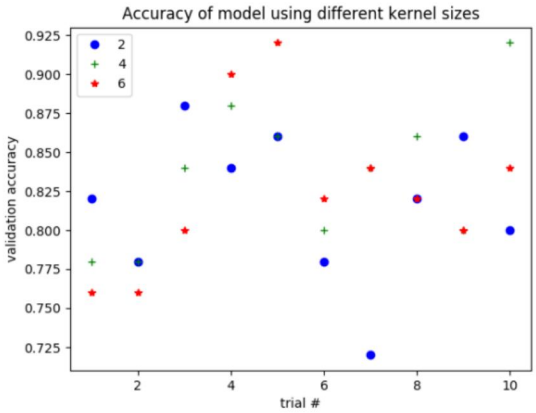
Since this experiment attempts to classify inputs into a binary output, a binary cross entropy loss function was used to evaluate the performance of both models:

$$\text{Loss} = -(y * log(p) + (1 - y)\log(1 - p))$$

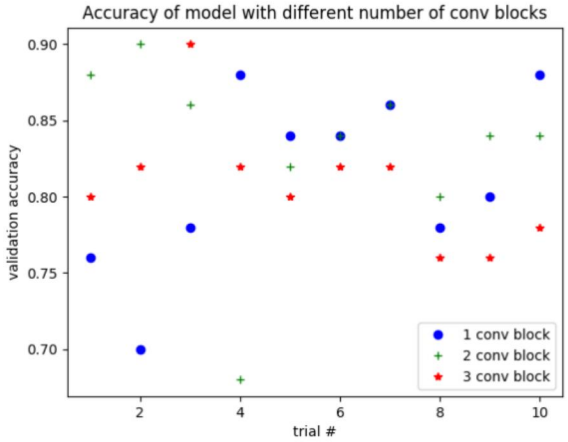
In the simple neural network architecture consisting of only fully connected layers, the main hyperparameters that were adjusted were the number of layers in the network, the number of nodes at each layer, and the dropout keep probability used for reducing overfitting. Because of the small number of examples available for training, having a very deep network did not make much sense as it can be prone to overfitting. It was empirically determined that using a neural network with 2 hidden layers was a reasonable balance between the amount of time it takes to train the network and the ability of the network to produce a high classification accuracy, since adding more hidden layers beyond the 2 hidden layers did not result in a better performing network. The number of hidden nodes in each layer was empirically chosen to be 50, and tuning this number did not produce a noticeable difference in model performance. The dropout keep probability was also tuned from a range of test values from 0.6 to 0.9, and interestingly, different dropout keep probabilities did not result in meaningful difference in model performance.

In the convolutional neural network architecture, the kernel size and the number of filters used in each convolutional layer were parameters that were tuned. Here are the results of experimenting with tuning the 2 different parameters:





From the results of the experimentation, using different numbers of convolutional channels and kernel sizes does not appear to make a significant difference in model performance. Different number of convolutional network blocks were then used in attempt to find an optimal number, and the following chart shows the results of this experimentation:



It is still a bit unclear whether or not adding convolutional network blocks to the overall architecture results in a substantially better performing model, but from the results of the experimentation, it appears that having more layers of network blocks may potentially make the model less prone to misclassifying examples that are perhaps more difficult to classify correctly.

Regularization and Data Augmentation

Correcting for the issue of overfitting was a central concern in this experiment where a relatively small dataset was used for training. It was very easy to train either the simple fully connected network or the CNN based network to achieve nearly 100% accuracy on the training dataset, but the network would always run into the issue of overfitting.

To combat this issue for both network architectures, dropout was used when training the simple fully connected network model while batch normalization layers were added in between convolution and ReLU activation layers in the CNN based architecture. Even with the utilization of these 2 regularization techniques, performance of the 2 models did not appear to improve much, and it appears that overfitting is still preventing the models from classifying the validation dataset at a better accuracy rate.

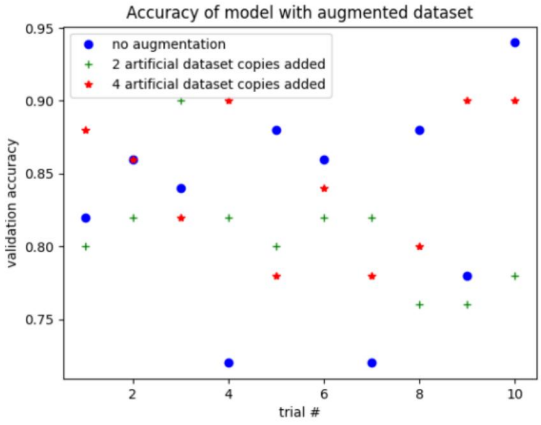
In order to resolve the central problem of the training dataset being too small, more time could have been spent to acquire more data for training. However, it would have been far too time consuming to collect more news articles, extract part of speech and text features from each article, and train the models with more data. Instead, a data augmentation technique where noise is added to an original copy of the training dataset and then the transformed copy of the dataset is added to the training set [3] is used in attempt to artificially augment the dataset.

In this data augmentation experiment, copies of the input feature values for each training example are made using the following technique:

$$\text{Artificially created input feature value} = \text{original feature value} + \text{rand}(-d, d)$$

A random decimal value between -d and d is added to the original feature value, with d chosen empirically.

Here are the results of experimenting with training the CNN based architecture with additional noise induced copies of the original training dataset:



One interesting takeaway from the experiment with adding artificial dataset is that it appears that overall classification accuracy of the validation dataset is a bit higher and less prone to poor classification accuracy of examples that are perhaps more difficult to classify when more noise induced copies of the dataset are added as additional training examples.

Here are the final hyperparameters used for each model and their respective validation dataset accuracies:

Simple neural Network	Dropout keep probability: 0.8	Number of hidden layers:2	Number of hidden nodes in 1st layer: 300	Number of hidden nodes in 2nd layer: 200	Validation accuracy: 83%
CNN based network	Number of CNN blocks: 2	Number of CNN filters used: (5 in the 1st layer, 6 in the 2nd layer)	Max pool layer parameters (pool size: 4, stride: 2)	Number of fully connected layers: 1	Validation accuracy: 84.6%

Conclusion

From the results of this experiment, there does not appear to be clear cut evidence of the superiority of either the simple neural network architecture or the convolutional neural network architecture in determining bias in sports news articles. Optimizing hyperparameters for either model also did not significantly improve classification accuracy. However, from the data augmentation experiment, it appears that perhaps by either artificially generating or collecting more training data to train the models, the convolutional neural network architecture may be able to generalize and classify unseen articles better. This provides hope that given more time to either further tune the data augmentation techniques or gather and parse more training data, a more accurate deep learning based model that takes text features as input can be developed and trained with higher classification accuracy. The difficult problem of bias detection in news articles can then essentially be broken down into 2 steps, first by extracting text features from news articles and then by passing extracted features into the aforementioned model.

Link to source code used in experiments in GitHub:

https://github.com/sqng629/cs230.git

References:
[1]: UCI Machine Learning Repository: Flags Data Set, 9 Apr. 2018, archive.ics.uci.edu/ml/datasets/Sports+articles+for+objectivity+analysis.
[2] Recasens, M., Danescu-Niculescu-Mizil, C., & Jurafsky, D. (2013). *Linguistic Models for Analyzing and Detecting Biased Language*. Retrieved from Association for Computational Linguistics website: https://nlp.stanford.edu/pubs/neutrality.pdf
[3] DeVries, T., & Taylor, G. W. (2017). *Dataset Augmentation in Feature Space*. Retrieved from ICLR website: https://openreview.net/forum?id=HJ9rLLcxg
[4] Doumit, S., & Minai, A. (2011). *Online News Media Bias Analysis using an LDA-NLP Approach*. Retrieved from https://eecs.ceas.uc.edu/~aminai/papers/doumit_iccs2011.pdf
[5] Hamborg, F., Meuschke, N., & Gipp, B. (2018). *Bias-aware news analysis using matrix-based news aggregation*. Retrieved from https://www.researchgate.net/publication/325228442_Bias-aware_news_analysis_using_matrix-based_news_aggregation
[6] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Retrieved from https://arxiv.org/pdf/1502.03167v3.pdf