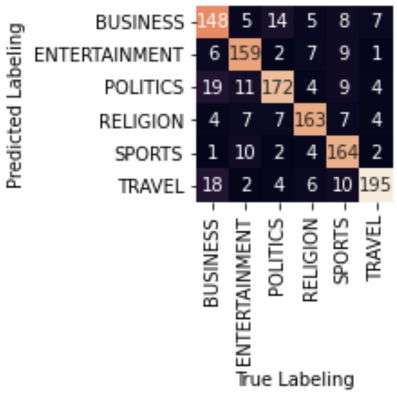
Project Title: News Category Classification

Team 1 Names: Benjamin Barnett and Yaniv Bronshtein

Data Source: https://www.kaggle.com/rmisra/news-category-dataset

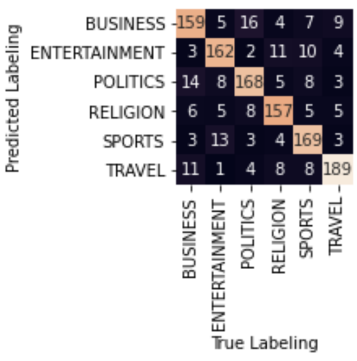
The goal of our project is to find models that can properly label untracked news articles. To do this, we applied deep learning to predict news article topics. Our dataset contains around 200k articles posted in 2012 to 2018 from HuffPost. We extracted these articles from Kaggle as a JSON file, and read the first 1000 articles from each category to work with a balanced dataset. Since each article includes a headline and a short description, we decided to train and test our models on the combined headline and description text. While we would have liked to evaluate the performance of these models on a total of 41 categories, we found that some categories overlapped. As an example, “Arts & Culture.” “Culture & Arts,” and “Arts” were a few of the labels. We decided to perform multinomial classification on 6 distinct categories: Politics, Entertainment, Travel, Business, Sports, and Religion. With 1000 articles for each topic, we used a grand total of 6000 articles for our combined training and testing data.

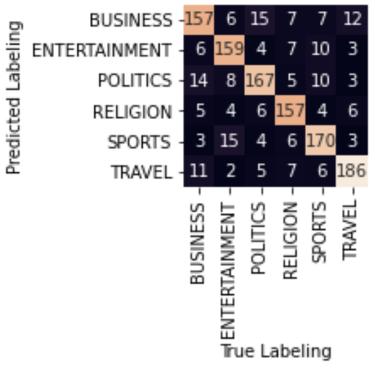
To divide our articles into training and testing data, we used the scikit-learn package and called *train_test_split* with *train_size* set to .8 and *random_state* set to 1. This created a random training and testing split with 4800 and 2200 articles each. Next, we applied the *TFidfVectorizer* function to make a pipeline with the Multinomial Naive Bayes classifier and fitted our model. Below is the confusion matrix for the predictions:

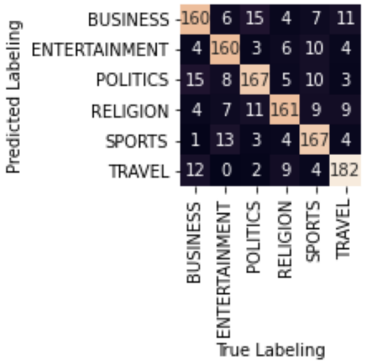


Given that the micro-averaged F1 score was calculated to be 0.834166666666666 and the macro-averaged F1 score was calculated to be 0.8333946355418081, the classifier does a good job in predicting the different categories. We decided that this would be an appropriate baseline to use given its solid classification performance.

Next, we applied the *TFidfVectorizer* function to make a pipeline with the *MLPClassifier*. Specifically, we decided to test a total of three multilayer perceptron networks, with 1, 2, and 3 hidden layers each with 100 neurons. The 'relu' activation function was utilized for the hidden layer, and the 'adam' solver was utilized for the weight optimization. We also used a constant learning rate for all three networks to maintain consistency. See below for their respective confusion matrices:





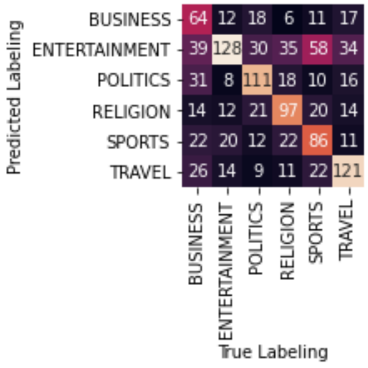


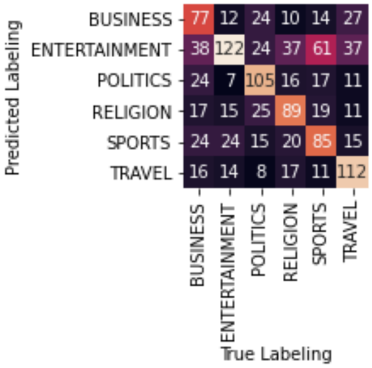
F1 Micro: 0.8366666666666667
F1 Macro: 0.8361764103490689

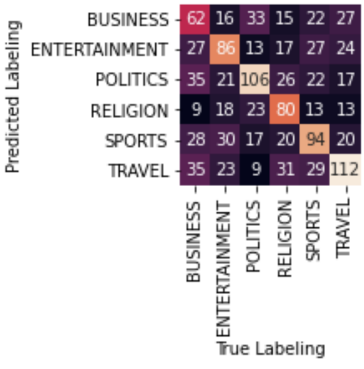
F1 Micro: 0.83
F1 Macro: 0.8297875759389398

F1 Micro: 0.8308333333333333
F1 Macro: 0.8306364902836441

As we can see, the classification performance for 1, 2, 3 hidden layers (read from left to right) is pretty much the same for all three. Furthermore, the classification performance is nearly identical to the performance of the Naive Bayes classifier. Interestingly, the first model had the largest number of iterations before stopping, with a total of 64 iterations, followed by 35 and 29 total iterations for the 2 and 3 hidden layer models, respectively. It seems that as more layers are added, fewer iterations are needed to reduce the loss. While these models all perform well using the TF-IDF matrix, we wanted to know if they would perform even better with embeddings. Removing stop words and assigning OOV stored random values within -1 and 1, we obtained the embedding results shown below:







F1 Micro: 0.5058333333333334
F1 Macro: 0.5023780402434700

F1 Micro: 0.49166666666666664
F1 Macro: 0.49184185045911420

F1 Micro: 0.45
F1 Macro: 0.44735404865164435

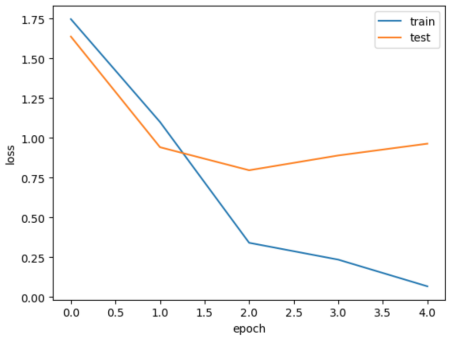
To our surprise, the embeddings performed much worse than we expected. All of the 3 models used the maximum number of iterations (set at 200), with low F1 scores. While the first two models had F1 scores at about 0.5, the model with 3 hidden layers had the lowest F1 scores at about 0.45. On the whole, we can see that MLP models with TF-IDF have much better performance than with the embeddings. One potential reason for this result could be that the TF-IDF approach is more appropriate for short text. Combining the title and short description of each article provides limited text, which likely prevents the embedding approach from performing well on testing data.

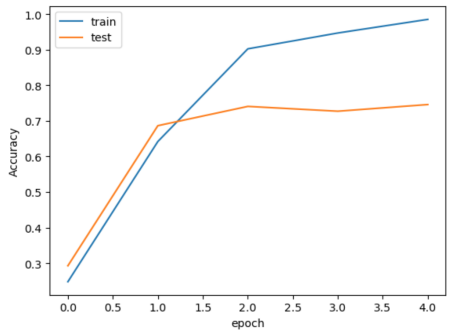
Given that the MLP models with embeddings have poor performance on combined title and short description text, we would like to evaluate their ability to classify articles using raw text. It would be interesting to see just how much improvement there would be with different input, and compare its performance to the other trained models. We expect that these will do much better than before, and potentially even better than the Naive Bayes classifier. While we have already utilized the BeautifulSoup library to parse out the text, we still need to clean the data before we can test the embedded models on the raw text. We may also want to implement other embedded models like RNNs and LSTMs on the raw text rather than only on the combined title and description.

We decided to train and test a LSTM model on the combined title and short description data. But after reading the documentation, we realized that the LSTM required a specific input format to maximize its performance. We wrote a custom function *clean_summary()* that used the following steps: lowercasing, replacing punctuation with spaces, replacing garbage characters with the empty string, and finally removing English stopwords. Next, we used the Tensorflow Tokenizer class to transform the new strings into a sequence of integers to feed into the model. For the labels, we did one hot encoding to transform the six categories into numeric features. We split the data into 60% training, 20% validation, and 20% testing (where the validation data was derived from the training data).

For the LSTM itself, we designed a Tensorflow Sequential() model with embeddings of dimension 100, and an input length of 250. In the first layer of our Sequential Keras, we added an embedding layer that used vectors of dimension 100 representing a maximum of 50,000 words. Next, a 1D Spatial Dropout layer was added with 20% dropout. For the LSTM layer, we had 100 neurons and 20% dropout to perform the bulk of the operation. Finally, we used a Dense layer with 6 nodes (corresponding to the 6 categories) and a softmax activation function. The loss function utilized was Categorical Cross Entropy, as we were not dealing with traditional binary classification. We also decided to implement the Adam optimizer with accuracy as a scoring metric as per common recommendation.

When the model was fit, a maximum of 50 epochs were allotted. However, the training never utilized more than 5 epochs due to the EarlyStopping callback introduced. There were 64 samples used for each batch, with 64 samples used for every gradient update. As the best case accuracy 0.778 (shown below), the results were good overall.





```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 250, 100)          5000000

 spatial_dropout1d_3 (Spatia  (None, 250, 100)         0
 lDropout1D)

 lstm_3 (LSTM)               (None, 100)               80400

 dense_3 (Dense)             (None, 6)                 606

=================================================================
Total params: 5,081,006
Trainable params: 5,081,006
Non-trainable params: 0
_________________________________________________________________
```

Summary of the LSTM model.