Categorizing News Articles with Manual Vectorization Methods and Pre-Trained Embeddings

Robin Singh

Northwestern University

Abstract:

For this assignment, I wanted to try and find some of the best ways in order to characterize the various news source data on their own. Previously, we had tried vectorizing the News_Categories_v2 dataset using methods such as Count vectorization, TF-IDF, and Doc2Ven, and then tried to characterize them using a Random Forest Classifier. Afterwards we tried clustering the data using K-Means, t-SNE, and bi-clustering. These did not perform as well as I had expected them to, as they had around a 56% accuracy at the best. So, I wanted to try some alternative motives in order to significantly improve the categorization accuracy through the use of neural networks. I still analyzed the best vectorization methods as I had done previously which were the TF-IDF and Doc2Vec. However, since they did not prove as good as a result as I thought it would, I also tried a pretrained embedding to see how much of an improvement that would make.

Introduction:

The reason for this experiment was to improve the categorization accuracy of the News_Categories_v2 dataset, as it was performing quite poorly in the previous assignments. I wanted to see how much I could bring up the predictive accuracy using alternative neural network methods to train and fit models. I used various different kinds of Neural Networks through TensorFlow to try and improve my predictive accuracy for the dataset, categorizing into five different categories which were World News, Sports, Money, Women, and Entertainment. The management problem that this experiment could solve is that it could assist a company or organization in finding the best way in which to categorize documents on their own based on the text within and without manually looking at the contents within their files, or for the organization to organize existing databases into more or less categories when necessary.

Literature Review:

Similar projects to this have been done by many people, such as an individual named Muneed, who created a blog about his project on DigitalVidya. He had created a text classification model, analyzing multiple techniques to classify the model, including Random Forest, Multinomial NB, and Logistic Regression. He had also clustered his data using K-Means clustering to see similar documents plotted together. After preprocessing all of the data and vectorizing his documents, he found the logistic Regression appeared to classify the best in his example.

Another example is that of Nikolai Janakiev, who conducted a sentiment analysis on data from IMDB, Amazon, and Yelp on order to predict whether or not a review was positive or negative. He used the Count Vectorizer Method in order to vectorize his data, and manually embedded the data in his RNN in order to perform predictive analysis on his data using binary cross entropy for his loss due to him only analyzing between positive and negative results.

Methods:

I began my experiment by preprocessing the dataset into more useable characters. This meant combining the tile and text summary together to conduct the analysis on, cleaning stop words, removing punctuation, and conducting Porter Stemming on the text prior to vectorizing them. Then, I had altered the data to only analyze 5 of the categories the news dataset had due to constraints in time and computing power. The five categories were World News, Sports, Money, Women, and Entertainment. I had split my data into train and test data, with 75% in the train and 25% in the test set. Afterwards, I began comparing the two techniques for vectorizing the data, TF-IDF and Doc2Vec. In order to perform the categorical cross entropy loss analysis on my data, I had to perform label encoding on the categorical outputs that I will be trying to predict. This

created a binary array of a length of 5 to determine which category the text belonged to. The first neural network I ran was the convolutional neural network. I created this CNN as so:

```
max_features =29535
embedding_dim =256


model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(2, embedding_dim,
                                    embeddings_regularizer = regularizers.l2(0.0005)))

model.add(tf.keras.layers.Conv1D(128,3, activation='relu',\
                                 kernel_regularizer = regularizers.l2(0.0005),\
                                 bias_regularizer = regularizers.l2(0.0005)))


model.add(tf.keras.layers.GlobalMaxPooling1D())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(5, activation='sigmoid',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))

model.summary()


model.summary()
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), optimizer='Nadam', metrics=["CategoricalAccuracy"])
```

The max_features represented the number of various words in the dataset, while the embedding_dim represented how large of a vector I wanted each text entry to be. I chose a large value thinking it would be able to retain more of the data that way due to the very unique nature of the text data.

A similar setup was used for the Doc2Vec vectorization method as well. For the Doc2Vec, however, negative values were not allowed in the matrix being fit through the CNN, so I had to conduct Min-Max scaling on the matrix beforehand in order for the data to be processed. Here is the architecture of the Doc2Vec CNN:

```
max_features =200
embedding_dim =200
sequence_length = 201

model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(max_features +1, embedding_dim,
                                    embeddings_regularizer = regularizers.l2(0.0005)))

model.add(tf.keras.layers.Conv1D(360,3, activation='relu',\
                                 kernel_regularizer = regularizers.l2(0.0005),\
                                 bias_regularizer = regularizers.l2(0.0005)))


model.add(tf.keras.layers.GlobalMaxPooling1D())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(4, activation='sigmoid',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))



model.summary()
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), optimizer='Nadam', metrics=["CategoricalAccuracy"])
```

This version had a smaller max_features value due to the Doc2Vec already vectorizing based on similarities into a vector of length 200 each.

Next was using a recurrent neural network to conduct predictive analysis. For the RNN, instead of using the TF-IDF vectorizing method, I had conducted count vectorizing by manually counting the amount of each word's occurrence and assigning them a value based on their index of the count matrix, padding the individual vectors to make them the same length. The Doc2Vec preprocessing was the same, however.

Here is the count vectorizer RNN architecture:

```
model=Sequential()

model.add(Embedding(num_words,64,input_length=max_length))
model.add(LSTM(128,dropout=0.1))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(5,activation='sigmoid'))
optimizer=Adam(learning_rate=3e-4)
model.compile(loss='categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
```

Here is the Doc2Vec RNN architecture:

```
model=Sequential()

model.add(Embedding(2,64,input_length=vec_size))
model.add(LSTM(128,dropout=0.1))
model.add(Dense(5,activation='sigmoid'))
optimizer=Adam(learning_rate=3e-4)
model.compile(loss='categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
```

Finally, I had chosen to run the text data through a pretrained embedding layer based on the 150gb Google News database, vectorizing my text according to that pretrained embedding. Then I had run a regular neural network, a CNN, and an RNN with the Google News hub layer.

NN Architecture:

```
model=tf.keras.Sequential()
model.add(hub_layer)
model.add(tf.keras.layers.Dense(16,activation='relu'))
model.add(tf.keras.layers.Dense(32,activation='relu'))
model.add(tf.keras.layers.Dense(16,activation='relu'))
model.add(tf.keras.layers.Dense(5,activation='sigmoid'))
model.summary()
```

CNN Architecture:

```
max_features =29535
embedding_dim =256


model = tf.keras.Sequential()

model.add(hub_layer)
model.add(Reshape((1,20)))
model.add(tf.keras.layers.Conv1D(128,1, activation='relu',\
                                 kernel_regularizer = regularizers.l2(0.0005),\
                                 bias_regularizer = regularizers.l2(0.0005)))


model.add(tf.keras.layers.GlobalMaxPooling1D())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(5, activation='sigmoid',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001)))



model.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), optimizer='Nadam', metrics=["CategoricalAccuracy"
model.summary()
```

RNN Architecture:

```
model=tf.keras.Sequential()

model.add(LSTM(120,input_shape=(1, 20)))
model.add(tf.keras.layers.Dense(16,activation='relu'))
model.add(tf.keras.layers.Dense(5,activation='sigmoid'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

This concluded the various neural networks I had conducted my analysis with.

Results:

The neural networks conducted with TF-IDF vectorization and Doc2Vec vectorization did not perform well. The first CNN conducted with the text vectorized through TF-IDF had received and accuracy of 33% on the training data and an accuracy of around 30% with the test data. The Doc2Vec method of vectorizing had a similar result. The RNN using the count vectorizer and Doc2Vec methods was the same story, with and accuracy of 34% on the training data and 31% on the test data.

Finally, we come to using the pretrained embedding. The regular neural network had an accuracy of almost 96% on the training data and almost 74% on the test data in only 20 epochs. The RNN CNN also both had an accuracy of similar numbers. This means that with such a unique dataset as these news text entries, a pretrained embedding may be required to get better results when categorizing into 5 or more categories.

Conclusion:

At first, I was very concerned as to why the accuracy was not improving in my various neural network models. I had done a lot of hyperparameter tuning and even removed some preprocessing steps in order to try and get a better accuracy when running the CNN and RNN on the count vector, TF-IDF vector, and Doc2Vec vector. However, it seemed that the model was not learning much, with the accuracy values not improving despite increasing the number of epochs I had ran it through. Perhaps, the data was just too complex and had lots of unique words which caused difficulty in learning relationships within the neural networks.

That's when the idea of using a pretrained embedding came up, to try and see if I can get better accuracy that way, and it worked significantly better. Using the embedding layer made from 150gb of Google News data created a new way to vectorize my text and perhaps combined a lot of similar words into similar numerical values. This allowed the neural networks to find correlations more easily and improve the accuracy of the models overall. Perhaps the differences made by the LTSM and 1-dimentional Convolutional layers may have become more apparent as the neural network ran for a longer time. This experiment shows us an efficient way in order to conduct document classification which could be useful in many use cases such as in search engines and in categorizing existing databases of records for better resource allocation and to easily find what one may be looking for.

Works Cited:

Muneeb. “Document Classification Using Python And Machine Learning.” Digital Vidya, 9 Jan. 2020, www.digitalvidya.com/blog/document-classification-python-machine-learning/.

Real Python. “Practical Text Classification With Python and Keras.” Real Python. Real Python, May 8, 2021. https://realpython.com/python-keras-text-classification/ .

Appendix:

CNN result for TF-IDF vectorization method:

```
Epoch 3/3
339/339 [==============================] - 2415s 7s/step - loss: 1.5411 - categorical_accuracy: 0.3380 - val_loss: 1.5383 - val_categorical_accuracy: 0.3394
```

CNN result for Doc2Vec vectorization method:

```
Epoch 20/20
339/339 [==============================] - 44s 131ms/step - loss: 1.5337 - categorical_accuracy: 0.3403 - val_loss: 1.5371 - val_categorical_accuracy: 0.3325
```

RNN result for Count vectorization method:

```
Epoch 20/20
339/339 [==============================] - 53s 157ms/step - loss: 1.5271 - accuracy: 0.3440 - val_loss: 1.5377 - val_accuracy: 0.3317
```

RNN result for Doc2Vec vectorization method:

```
Epoch 20/20
339/339 [==============================] - 34s 100ms/step - loss: 1.5305 - accuracy: 0.3406 - val_loss: 1.5443 - val_accuracy: 0.3261
```

Basic NN result with pretrained embedding layer:

```
Epoch 20/20
339/339 [==============================] - 3s 8ms/step - loss: 0.1509 - accuracy: 0.9577 - val_loss: 1.2445 - val_accuracy: 0.7376
<tensorflow.python.keras.callbacks.History at 0x7fcfad522e90>
```

CNN result with pretrained embedding layer:

```
Epoch 20/20
339/339 [==============================] - 2s 6ms/step - loss: 0.2701 - categorical_accuracy: 0.9370 - val_loss: 1.0631 - val_categorical_accuracy: 0.7440
<tensorflow.python.keras.callbacks.History at 0x7fcfafc71990>
```

RNN result with pretrained embedding layer:

```
Epoch 20/20
339/339 [==============================] - 1s 4ms/step - loss: 0.1241 - accuracy: 0.9598 - val_loss: 1.5445 - val_accuracy: 0.7368
<tensorflow.python.keras.callbacks.History at 0x7fd0109b2d50>
```

Legend for Files:

Assign4CNN.ipynb is the CNN using the TF-IDF vectorization method.

Assign4CNNDoc2Vec.ipynb is the CNN using the Doc2Vec vectorization method.

Assign4RNNLTSM.ipynb is the RNN using the TF-IDF vectorization method.

Assign4RNNLTSMDoc2Vec.ipynb is the RNN using the Doc2Vec vectorization method.

(Poor models; 34% accuracy)

****

Pre-trained Embedding Assignment4.ipynb is the basic NN, CNN, and RNN with the Google News 150gb pre-trained Word2Vec embedding layer

(Good Models; over 95% accuracy)