Machine Learners Final Project

SENTIMENTAL ANALYSIS

Justin Cote, Sandip Kushwaha, Amal Ronak, Tony Zhang

Github Repository URL : FinalProject448

Libraries Used: pandas, numpy, tensorflow

We learned a lot on how to build these models using the following documentation :
Keras Docs and Kaggle Website Documentation.

Task and Dataset preprocessing

We retrieved our dataset from kaggleDataset. The dataset was structured by two columns: Clean_comment and Category. The Category column was restricted to three of the following values; [-1(negative) ,0 (neutral), 1(positive)].

This is the column we will predict on so we extracted it for the training data. Clean_comment is the string value that corresponds to Category. This is the comment we need to extract to train the model. Thus we extracted it and later prepared it to be used correctly when training the model. After unzipping data to our current file, we cleaned the data to remove special characters. To prepare the data we filled any null spaces with an empty string and encoded our text data and sentiment labels to numerical representations using a Tokenizer. Once all data was converted to numerical values, we could split the data into sets; train and test.

Implementation and architectures

CNN:

We implemented our Convolutional Neural Network (CNN) using the TensorFlow and Keras frameworks. The implementation is encapsulated within the CNNModel class, providing a structured and modular approach to building and training the model.

The architecture, encapsulated within the CNNModel class, encompasses key components such as an embedding layer for vectorizing words, a 1D convolutional layer with ReLU activation, global max-pooling for feature extraction, and dense layers facilitating sentiment classification into negative, neutral, and positive categories. The model is trained with categorical cross entropy loss, Adam optimizer, and employs early stopping to prevent overfitting during the training process. This CNN architecture is tailored to capture hierarchical features in text data, particularly for sentiment analysis.

RNN:

The model architecture begins with an embedding layer, mapping words to vectors, followed by a SpatialDropout1D layer to introduce dropout in the embedding space. The core of the model is an LSTM (Long Short-Term Memory) layer with 100 units, incorporating dropout and recurrent dropout for regularization. The final layer is a dense layer with softmax activation, indicating the model's classification into three sentiment classes: negative, neutral, and positive. During training, early stopping is implemented to monitor the validation loss, and training is halted if improvement ceases, restoring the best weights.

Training Details

The dataset is loaded from a CSV file and preprocessed by cleaning the text data and filtering relevant sentiment labels. The preprocessing steps include transforming text data into sequences of numerical values and encoding the sentiment labels. The dataset is then split into training, validation, and test sets using the train_test_split function from scikit-learn. The training set comprises the majority of the data, with the validation set used for monitoring the model's performance during training and early stopping to prevent overfitting. The test set remains unseen during training and serves as an independent evaluation set to assess the model's generalization to new data. We split our data so that we had 97783 training samples to 32595 test and validation samples. This provided a larger test case to train the model on so that our accuracy would be increased after.

Results and Observations

The CNN model exhibits a strong training performance, reaching a training accuracy of 98.91% after five epochs, while the validation accuracy stabilizes at 95.75%. The test accuracy is consistent with the validation accuracy at 95.69%, showcasing the model's generalization to unseen data.

On the other hand, the RNN model demonstrates a comparable training accuracy of 98.18% after five epochs, with a slightly higher validation accuracy of 96.79%. The test accuracy is consistent with the validation accuracy at 96.85%. The RNN model seems to capture sequential dependencies effectively, leading to a slightly higher validation and test accuracy compared to the CNN model.

In conclusion, both models exhibit strong performance in sentiment analysis, with the RNN model marginally outperforming the CNN model in terms of validation and test accuracy. The choice between these models may depend on specific use cases and preferences, considering factors such as computational resources and interpretability.

Challenges And Obstacles

Our first major obstacle was an issue with our dataset. At first we wanted to do something with stock data so we tried to predict stocks with CNN and RNN which did not work too well. After reviewing the slides and lectures we found a more optimized solution. Hence our chosen task being sentimental analysis on Twitter comments. We found that adjusting our dataset for the given deep learning algorithms affected our output positively.

For cleaning the datasets, we encountered obstacles due to data that fell outside the defined labels. Eliminating these mislabeled or erroneous entries was crucial to ensure the integrity of our dataset. It required meticulous scrutiny and filtering processes to remove these discrepancies and maintain the quality of the data we were working with.

Moreover, during the testing phase of our code, we confronted issues related to errors and libraries. Installing each required library on our machine became a necessary step, and we meticulously checked the functionality and compatibility of each code snippet. Debugging was a significant part of this phase, ensuring that the code executed seamlessly without any errors.