Abstract—

I. Method and Materials

A sample of texts is required to classify the binary sentiment of a movie script or Twitter Tweets. The movie script text samples represent dimensional space for the set task. The words in the language determine the sentiment of the text sample. A supervised machine learning algorithm, a deep neural network known as a C-LSTM, which is a hybrid neural network including convolutional and long-short-term-memory layers, makes it possible to extract the sentiment from the text alone. Applying adaptable parameters and structures within the C-LSTM layers provides an architecture capable of performing text classification tasks reliably across different dimensions of texts. All modeling efforts are captured in the following order:

1) Text Pre-Processing & Sentiment Labeling
2) Hyper-Tuning of the PTCN
3) Best Parameters PTCN Model 5-fold Cross-validation
4) Capture Embedding Features from the Predictions

A series of natural language processing techniques are required to supervise the training of the PTCN to classify the text sentiment. The text samples contain noise, which is in the form of stop words, uppercasing, special characters, NA values, punctuation, numbers, and whitespace. The removal of noise from the text samples helps mitigate the C-LSTM, recognizing patterns within the texts that create bias predictions. Text samples with mitigated noise are not the only requirement for training the PTCN. A classification of each of the text samples' binary sentiment is needed. A label of sentiment for each text sample is completed by calculating the polarity of each word in a text sample and concluding the overall polarity score. The overall polarity score determines the classification of sentiment for each text based on a conditional threshold of above or below a neutral value of 0. If the polarity is above 0, the classification label is positive for that text sample. If the value is below 0, the classification label is negative. Each polarity measure is converted into a binary label depending on the conditional threshold.

The PTCN is a deep learning algorithm that automatically extracts features from an input vector with adaptable kernel initializers. An additional batch normalization layer and dropout layer are implemented to help avoid bias in the model's predictions. Deep neural networks transform inputs over extensive credit assignment pathways (CAPs). The more complicated the dimensions of input, then the more complex the CAPs. The labeled sentiment text samples are balanced into an equal distribution to confirm that the features extracted throughout the transformation of the text inputs are significant. The C-LSTM will be trained on an equal amount of random positive sentiment text samples and random negative sentiment text samples. For the hyper-tuning model, the text samples are embedded into the layers of the C-LSTM in the form of a tokenized word vector. The space of the tokenized word vector is the representation of the positive and sentiment movie scripts that the model will extract features to form an accurate prediction. A list of input samples is tokenized in sequence to form numerical values for each represented word across the texts. Each text is transformed into a numerical sequence of tokenized words in the form of a vector. The newly formed word vector spaces are attached to their designated sentiment label. The lists of input vectors and output scalars require further transformation to embed with the C-LSTM. The text inputs are padded in order to make each text the same size. The list of padded inputs is transformed into a matrix. The outputs are converted into categorical variables for each sample transforming from a list of scalars into a matrix of binary responses. The complete sample of text and sentiment labels are split for training and evaluation datasets.

For the best model, a small variation is required in the processing of the sentiment labeled text samples. Texts processing techniques all remain the same, but the sampling for training and validation sets is changed to incorporate 5-fold cross-validation. A fixed set of 100 samples is reserved for evaluating the best model's folds. After preprocessing the data for the hyper-tuning and best model, the data is embedded into the C-LSTM layers and transformed through the stacks. The inputs are embedded based on their dimensions, including the max features, embedding dimensions, and max length of the inputs. The inputs are piped through a series of layers in the model. 1-dimensional convolutional layers are set with a series of elements to control various parameters of the convolutions. The input samples are large vectors with high sparsity requiring that the layers be allowed to explore but in a controlled manner. A series of convolutional layers are stacked and activated using a leaky Relu function to promote the model identifying patterns in the inputs. The model does not overfit because max-pooling layers are implemented at specific points in the architecture of the model. To further mitigate overfitting. The model is designed to pipe the inputs through a series of convolutional layers and then into a long-short-term-memory (LSTM) layer set with various parameter controls and activated with a leaky Relu function. In the last layer, the inputs are piped into a fully connected layer activated using a sigmoid function. The model is compiled using a binary cross-entropy function, optimized with a nadam function, and reporting results in set metrics.

The model is fit to the training dataset of inputs and outputs through a set number of epochs. Deep learning models validate after each epoch. A specific percentage of the training data is set for validation. To ensure the model does not train further, then it can learn, early stopping parameters are utilized to automate a change in the learning rate of the optimizer function and stop the model if improvement plateaus after a set number of epochs. The best weights are captured and set for the final model state that is saved for evaluation. The evaluation dataset is tested using the fitted model and determines the model's ability to generalize to new inputs. Hyper-tuning the model's parameters will ensure possible values to set for final modeling. The final model is trained using 5-fold cross-validation to fully understand the predictive capabilities of the model.

To ensure that the model is understanding the language after the best model is captured and cross-validated. The features embedded in the layers of the network are visualized to express the features of words that support or contradict the predicted sentiment classification. Understanding the features helps determine the usefulness of the convolutions recognizing features from the word vector spaces. The layers throughout the stack our essential to the performance of the PCTN. The sequential deep learning model is constructed with a stack of different layers set with a number of parameters, including:

- Dropout rate,
- Convolutional hidden nodes,
- LSTM hidden nodes,
- L1 regularization rate,
- L2 regularization rate,
- Batch size,
- Input max length,
- Max features,
- Embedding dimensions,
- Leaky Relu rate,
- Kernel size,
- Epochs,
- Max-pooling size,
- Learning rate, and
- Validation split.

The first layer in the model is the embedding layer, which embeds the tokenized words into the network. The word vector spaces reflect a commonly referred to method in natural language processing known as a bag of words method. The convolutional and LSTM layers in the architecture are capable of extracting features automatically from the highly sparse dimensional space of the processed inputs. The next layer the inputs are piped into is a 1-dimensional convolutional layer with only a 4th of the set convolutional hidden nodes. The inputs are representations of long and highly sparse word vector spaces which makes the initialization of the model weights critically important. The model requires a method to adjust to the high variance between samples. The control of the kernels will help the model steadily learn from the features extracted from the text.

Using a variance scaling kernel initializer provides an "… initializer that is capable of adapting its scale to the shape of weights." (R-Documentation, 2019). The Convolutional layers initializer make the deep network adapt to the input weights. It is important to regularize the kernels when utilizing an adaptive initializer. Setting a duel regularization technique, which deploys L1 and L2 regularization helps mitigate high fluctuations while batching samples through the layers. L1 is Lasso Regression (LR). It is a penalty term measuring the ". absolute value of the magnitude of the coefficient." (R Documentation, 2019). L2 is Ridge Regression. A penalty term measuring the "… squared magnitude of coefficient." (R Documentation, 2019). The main difference between the two methods is the penalty term. The 1-dimensional convolutional layers are set to inputs padded to the same shape. Strides are set to 1L through the convolutions, which are an "… an integer or list of a single integer, specifying the stride length of the convolution. (R Documentation, 2019). The convolutional layer is activated using a Leaky Relu function. The leaky rely function allows for a ".small gradient when the unit is not active", providing "… slightly higher flexibility to the model than traditional Relu." (R Documentation, 2019). The first convolutional layer extracts lower level features from the inputs due to the decrease in the hidden nodes. The reduction of the number of transformations provides control to the adaptable features initializing the weights. A high kernel size provides the layer with the ability to transform the data at a higher level through the few transformations. The second layer piped to is another 1-dimensional convolutional layer with the same parameters set but except the number of hidden nodes is set to 32. Increasing the number of hidden nodes provides more transformations extracting higher level features from the inputs. The second convolutional layer is activated using the leaky Relu function. The next layer in the stack is another convolutional layer set at half the set number of hidden nodes.

To mitigate overfitting a max-pooling layer set to 4 is implemented. Following is two more layers of 1-dimensional convolutional layers set to half the number of set hidden nodes and a 4th of the set hidden nodes. All parameters are set the same as the prior convolutional layers. A second max-pooling layer, batch normalization, and dropout layer are set to further mitigate overfitting. The next layer is a LSTM layer with the set at 32 hidden nodes. Variance Scaling kernel initializers and L1 and L2 kernel regularizers are implemented. The LSTM layer is activated using a Leaky Relu function. A dropout layer set at 0.5 is implemented before the output layer. The output layer is activated using a sigmoid function. The model is compiled using a loss function of binary cross-entropy. The stochastic gradient descent (SGD) optimizer is set with a learning rate at a specific learning rate. The learning rate will be adjusted to different rates during the hyper-tuning sessions to provide the best rate of learning for the final model. The models will provide training, validation, and evaluation metrics in form of accuracy percentage. The results will also provide the loss measure of the training, validation, and evaluation.

The embedding layers help understand the capabilities of the model. The model's predictions can be explained and provide insight into the features supporting or contradicting the sentiment classification of a text sample based on the weighted measure. The results include the predicted samples label, probability, and explanation fit. Within the visual 20 features are set to express their weights to support or contradict the texts classification.