CP8305 Knowledge Discovery

By: Mushahid Khan

November 23, 2020

1. Introduction

For the final project I decided to work on a programming project where I built two text classification models for Twitter sentiment analysis. The first one is an Artificial Neural Network model and the second one is a Recurrent Neural Network model. Sentiment analysis is extremely important for anyone using it, such as for businesses wanting to detect sentiment in customer feedback. It is extremely useful in social media monitoring as it enables us to obtain an overview of the public's opinions behind topics.

2. Data

For this project, I decided to use a dataset that I found on Kaggle called Twitter US Airline Sentiment. This dataset contains people's reviews of different airlines using Twitter tweets. The file is called 'tweets.csv' and it contains 14,640 instances and columns features for each instance. Of all the 15 columns, only two are important for this project, namely the actual tweet and the sentiment of each tweet. Each sentiment is either positive, neutral or negative.

2.1 Data Pre-processing

Data pre-processing involves getting the data ready for the learning algorithms. For this project, it involved selecting necessary features that can be useful in terms of predicting the target, removing unnecessary text from instances of selected features, checking for missing values, splitting the data into training as well as test data sets, converting target classes into vectors and applying word embedding on selected features. Because the dataset was available on a csv file, using Pandas library was enough to read in the csv file.

2.1.1 Feature Selection

The first task of data pre-processing that had to be done was feature selection. This is the process of selecting those features that would contribute the most to predicting sentiment on tweets. It is important to have relevant features to increase accuracy of the model. The dataset used in this project has 15 columns representing features. Out of the 15 columns, the actual tweet of the user, called text was selected as an attribute and the sentiment of the tweet, called airline_sentiment, was selected as the target class of each instance. Using Pandas library, I was able to keep only the two mentioned columns in the dataset. The first 5 instances of the dataset can be seen below in figure 1 after feature selection was applied.

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative
...	...	...
14635	@AmericanAir thank you we got on a different f...	positive
14636	@AmericanAir leaving over 20 minutes Late Flig...	negative
14637	@AmericanAir Please bring American Airlines to...	neutral
14638	@AmericanAir you have my money, you change my ...	negative
14639	@AmericanAir we have 8 ppl so we need 2 know h...	neutral

Figure 1: Dataset after feature selection is applied

2.1.2 Check for Missing Values

Datasets can contain missing values. To deal with missing values, we can drop the rows that have missing values. To deal with this, we can use the Pandas library. Luckily, this dataset did not have any missing values. There are 9,178 negative tweets, 3,099 neutral tweets and 2,363 positive tweets.

2.1.2 Remove Unnecessary characters

Each Twitter tweet contains a mention, preceded by @. The mention is a Twitter username. For sentiment classification, mentions are not needed as they do not add value and so they can be removed from the text column of each instance. Also, there are words called stop words that can be filtered out of the tweets. Such words do not add much value for a sentence. Examples of such words are the, is, which and on. Such words, with the exception of “n’t”, “not” and “no” as they provide value for sentiment classification, have been removed using Natural Language Toolkit (NLTK), an open source Python library for Natural Language Processing. Words of length 1 have also been removed.

2.1.3 Split the Dataset into Training and Test Sets

To split the dataset, I decided to allot 80% of the data to the training set and 20% to the test set. This was done by using Scikit-Learn library’s train_test_split method. Scikit-Learn library’s train_test_split method enables us to also separate the features from output labels. So, we get four subsets from the dataset which are training data feature set, consisting of 80% of the text feature, training data output labels, test data feature set, consisting of the remaining 20% of the text feature as well as test data output label. The four subsets are called X_train, y_train, X_test and y_test. When training of the models takes place, the training data will further be split into a validation data set to assess the model before testing it and a data set to do the training only. The training data has 11,712 instances and the test data has 2,928 instances.

2.1.4 Convert Target Class to Matrix Class

Because the y_train and y_test are of type string being either “Neutral”, “Positive”, or “Negative”, they have to be turned into vectors. To turn each target class into a vector, the LabelEncoder class from the Scikit-Learn

library was used. Here, each element of y_train and y_test will become a vector of 3 rows and 1 column where an entry of 1 in the first row corresponds to negative, entry of 1 in the second row corresponds to neutral and entry of 1 in the third row corresponds to positive sentiment. All other entries in the vector will be 0.

2.1.5 Word Embedding

Word embedding is the mathematical representation of text as vectors of integers. Each tweet in the dataset has to be converted into a vector of integers. Here, each word of the tweet will be represented as an integer. This form of representation will allow similar words to have a similar representation. This allows us to vectorize a text corpus by turning text into a vector of integers using the number of unique words we have in our dataset of tweets. For word embedding, Keras library's Tokenizer class was used in this project. All of the tokenized vectors need to have equal length and so some vectors may need a padding of the integer 0 to be of the length of the vector that represents the tweet of maximum length from the entire data set, which is 38. For this, Keras library's pad_sequence method was used. The tweets have to be vectorized so that they can be used as inputs in the Neural Network and Recurrent Neural Network.

3. Models

For this project, I wanted to see the effectiveness of deep learning in performing sentiment analysis. Deep learning models are neural networks inspired by the human brain. For different tasks, there exist different types of neural networks.

3.1 Artificial Neural Network

An Artificial Neural Network(ANN) is a collection of connected nodes. An ANN consists of input, hidden and output layers. Each layer has units that transform the input into an output that can be used in the next layer.

Below in figure 2, we can see a simple Artificial Neural Network with an input, hidden and output layer. Each node in a layer is connected to all nodes in the next layer with a certain weight. In ANN, each training example will pass through this network.

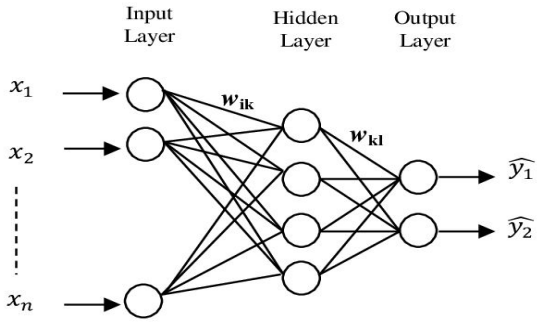

Figure 2: Artificial Neural Network

For creating the ANN, I used the Keras library from Tensorflow. An ANN model can be created using Keras's Sequential API. The Sequential API enables the creation of models layer by layer. For the ANN model, I decided to create the following layers: an input layer, an embedding layer, a flatten layer and a dense layer, as shown in the figure 3 below.

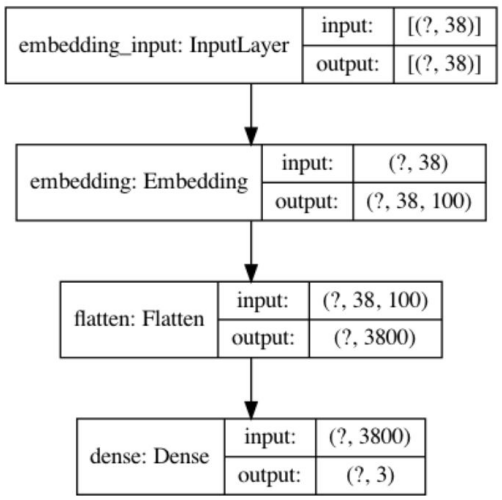

Figure 3: ANN used for this project

3.1.1 Input Layer

The number of nodes in the first layer, input layer, is equal to the length of each vector after word embedding took place on each tweet in the training set. This length is 38. This layer will take in all the vectorized tweets and pass them along to the embedding layer. The shape of output of this layer will be (?, 38) where ? is the number of tweets in the training set.

3.1.2 Embedding Layer

The second layer in the ANN is called the Embedding Layer. When word embedding was applied in the data pre-processing stage, I was able to get the contents of each tweet. However, I needed to get the context of each tweet as well. For example, consider the two phrases and the result of them going through word embedding described above in the data pre-processing stage:

- Have a good day - [0, 1, 2, 3]
- Have a great day - [0, 1, 4, 3]

As humans, we would be able to read the two phrases and know that the two phrases mean the same thing. Looking at the vector representation of the two sentences, however, all we can tell is that the words at index 2 are different. In other words, looking at the vector representations, we cannot tell how different the two phrases are. This is where we can use the embedding layer. The embedding layer will enable us to map each word to a position in vector space of some dimension. In other words, similar words are grouped together. For the embedding layer, the size of the embedding vector needs to be specified. After trying numerous values, the best result was obtained from 100. In other words, the vectorized tweets had to be mapped to the dimension of 38 by 100. The output of this layer is of the shape 38 by 100 for each tweet.

3.1.3 Flatten Layer

In the flatten layer, the output from the embedding layer will be flattened to vectors of dimension 3800. In other words, this layer unrolls the value obtained from the embedding layer into one vector. There will be 3800 nodes in this layer. This is similar to the hidden layer in figure 4.

3.1.4 Dense Layer

The dense layer acts as the output layer. It implements the operation of activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer. For the dense layer, the number of nodes and the activation function need to be specified. In this case, since there are three sentiments the number of nodes in the dense layer will be 3. The activation function used is the softmax function. The dense layer returns an output of probabilities for each sentiment. The sentiment with the highest probability is chosen for the given tweet.

3.2 Recurrent Neural Network

Recurrent neural network(RNN) is a type of deep learning algorithm commonly used in natural language processing. RNN makes use of sequential information. In the NN model, all inputs as well as outputs are independent of each other. RNN, on the other hand, performs tasks with the output being dependent on the previous computations. For this project, I used a special kind of RNN called Long short-term memory(LSTM). To implement this model, I used Keras's Sequential API. Using Sequential, I created the following layers: an input layer, an embedding layer, two dropout layers, an LSTM layer and a dense layer as shown in the figure 4 below.

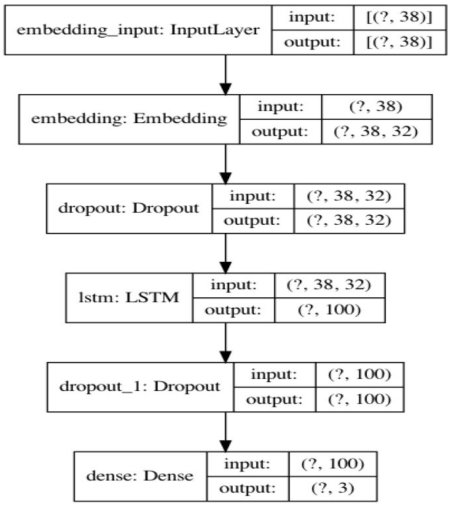

Figure 4: RNN used for this project

Before the vectorized tweets can be fed into the LSTM layer, they have to go through the embedding and dropout layers. For this model, having the output dimension for the embedding layer be 32 worked best.

3.2.2 Dropout Layer

Deep learning models can quickly overfit to a training dataset. The dropout layer ignores some nodes randomly during the training stage. At each training stage, a fraction of the input nodes are ignored in the network . Doing this allows the model to not be overfit to the training data. Given the dataset, the best results were obtained by adding a dropout layer right before the LSTM layer and right after it. For the first dropout layer, ignoring half of the input nodes and for the second dropout layer ignoring 20% of the input nodes works best here.

3.2.1 LSTM

Originally, RNN had the problem of vanishing gradient. Vanishing gradient is the issue found in training a neural network with back propagation. This problem makes it hard to learn the parameters of early layers in the network.

LSTM can tackle this problem by remembering historical data in memory easily. The output of this layer will return a single vector per training data.

3.3 Determining Number of Epochs

Apart from having the overall structure of the both models, I also had to figure out the best number of epochs. One epoch is when an entire dataset is passed through the neural network once. Having too many epochs can mean the model overfits the training data and can potentially perform badly on the validation as well as test data. To determine the best number of epochs, I decided to train both models with different numbers of epochs. I took the training dataset and split it further into two sets, one used to train the model and one which was used as the validation set. I decided to use 20 percent of the training dataset as the validation set. In figure 5, the training data accuracy as well as validation data accuracy can be seen for training NN on different numbers of epochs and in figure 6 the same can be seen for RNN.

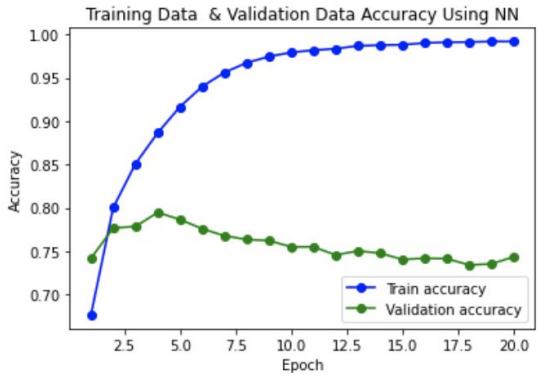

Figure 5: Accuracies using NN

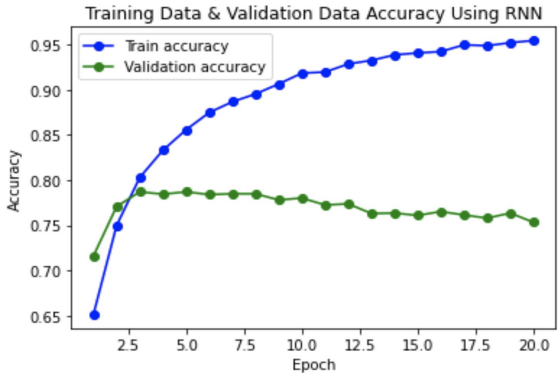

Figure 6: Accuracies using RNN

In figure 5 it can be seen that for the NN when the number of epochs go beyond 4, the validation data accuracy falls while the training data accuracy keeps rising. The same can be said for RNN beyond 3 epochs. This means that when the NN model is used, the model starts to overfit the training data after 4 epochs and

when the RNN model is used, the model starts to overfit the training data after 3 epochs. This lets us know that the NN model should be trained using 4 epochs and the RNN should be trained using 3 epochs.

4. Evaluation

To evaluate the models, I used the test set. Below in figure 7 is the confusion matrix for the NN model and in figure 8 is the confusion matrix for the RNN model using the test set. The confusion matrix compares the actual target values with those predicted by a model and gives a holistic view of how well the classification model performed.

		True Class		
		Negative	Neutral	Positive
Predicted Class	Negative	1702	93	41
	Neutral	239	303	59
	Positive	97	64	330

Figure 7: Confusion matrix for NN model

		True Class		
		Negative	Neutral	Positive
Predicted Class	Negative	1740	57	39
	Neutral	262	271	68
	Positive	97	58	336

Figure 8: Confusion matrix for RNN model

Using the confusion matrices of both models, I was able to calculate the accuracy, recall, precision as well as f1-score using both models. Accuracy is the fraction of predictions the model got correct, precision tells what proportion of the positive identifications of each class were actually correct, recall tells what proportion of actual positives of each class were identified correctly and f1-score is the harmonic mean of precision and recall of each class. The values of precision, recall and f1-score can be seen in table 1 below for the NN model and for the RNN model in table 2.

Class	Precision	Recall	F1-Score
Negative	0.845	0.927	0.879
Neutral	0.659	0.504	0.571
Positive	0.767	0.672	0.717

Table 1: Precision, recall and f1-score for the NN model

Class	Precision	Recall	F1-Score
Negative	0.848	0.928	0.886
Neutral	0.679	0.589	0.589
Positive	0.747	0.722	0.722

Table 2: Precision, recall and f1-score for the RNN model

Also, the NN model achieved an accuracy of 79.75% and the RNN model achieved an accuracy of 80.57%. Comparing the above results of evaluations for the two models, it can be concluded that the RNN model overall performed better than the NN model. This makes sense as the RNN model has more layers than the NN model as well as a more complex model than the NN model and as a result is a better model.

5. Execution of Models

In the deliverables, there is a Jupyter notebook called Execute.ipynb. To train and evaluate the above 2 models, the Execute.ipynb needs to be run. In this Jupyter notebook, the first cell will import and call the main method which will train and evaluate the NN model and the second cell will do the same for the RNN model. For the models to be executed in Jupyter notebook, the following libraries need to be installed:

- Tensorflow
- Numpy
- Seaborn

- Pandas
- Keras
- Matplotlib
- Nltk
- Scikit Learn

These libraries can also be seen in the Requirements.txt file.