Machine Learning Assignment 4 Hands on with ANN

(43 pts)

Problem1— Using ANN for Covid Sentiment classification

For this problem, you use the covid sentiment dataset you used in assignment 2 but instead of Naïve Bayes, you will use an ANN to classify covid tweets into "negative", "neutral" and "positive" sentiments.

Preparing Tweets

In module 5, we used the "tm" package for converting tweets into a document-term binary matrix where entry at row i and column j was "yes" if tweet i had word j, otherwise it was "no".

In this assignment we will use Keras instead of "tm" package for vectorising text and creating a document-term matrix from tweets.

1- First use "qdap" package to remove stop words and do stemming as follows (note: replace "covid" with whatever you called your dataframe)

```
install.packages("qdap")
library(qdap)

covid$OriginalTweet=rm_stopwords(covid$OriginalTweet, stopwords=tm::stopwords("english"),
separate=FALSE, strip=TRUE)
covid$OriginalTweet=stemmer(covid$OriginalTweet, warn=FALSE)
```

2- Randomize the order of rows, use the same seed as you did in assignment 2 so we can compare the two models on the same test dataset.

3- Similar to assignment2, Convert sentiment into a factor variable with three levels: "positive, "neutral", and "negative". Then convert this factor variable to a numeric vector (similar to how we encoded the labels for adult dataset in the lectures(slide 94))

4- Spit the data three ways in to train/validation/ and test sets as follows: use the first 26340 rows for training, next 6585 rows for validation, and the last 8232 rows for testing. Make sure that you are using the same test set as you used in assignment 2 so you can compare the ANN model with your naïve Bayes model in assignmentt2.

5- Keras has a preprocessing layer, called layer_text_vectorization, this layer creates a document-term matrix where rows represent tweets and columns represent terms. Use the following code segment to create document-term matrix for your training, validation and test datasets you created above. (Note: replace covid_train, covid_test, and covid_val with the names you gave to your train, test and validation sets):

```
library(keras)

text_vectorizer <- layer_text_vectorization(output_mode="tf_idf", ngrams =2, max_tokens =
5000)

text_vectorizer   %>%   adapt(covid_train$OriginalTweet)


covid_train_dtm, = text_vectorizer(covid_train$OriginalTweet)

covid_val_dtm =text_vectorizer(covid_val$OriginalTweet)

covid_test_dtm= text_vectorizer(covid_test$OriginalTweet)
```

layer_text_vectorization configures a text vectorization object. Let's discuss the parameters we passed to it:

- ngrams =2 means instead of tokenizing the text into single words, tokenize them into single words and two consecutive words. In general, word ngrams are groups of N (or fewer) consecutive words that you can extract from a sentence. Here's a simple example. Consider the sentence "gas prices are expensive" It may be decomposed into the following set of 2grams { "gas", "prices", "gas prices", "are", "prices are", "expensive", "are expensive"}
- max_tokens = 5000 Means only consider the top 5000 frequent tokens/bigrams as features/columns in the document term matrix and discard all other tokens
- output_mode="tf_idf" means fill the document-term matrix with tf-idf weights. In assignment 2, we filled our document-term matrix with "yes": and "no" representing absent or presence of a term in a tweet. However, knowing how many times a particular term occurred in a tweet can be very informative. For example, a tweet that contains the word "terrible" only once could be both negative or neutral but a tweet that contains the word "terrible" several times is most likely negative. At the other hand, some words like "covid" and "supermarket", "store" or "food" occur frequently in both positive, negative, and neutral tweets and they are not very informative for the classifier. In general, the more a given term appears in a document the more important that term is for understanding what the document is about. At the same time, the frequency at which the term appears across all documents in your dataset matters too. Terms that appear in most tweets (such as "covid", "supermarket", "food") aren't particularly informative but terms that appear in a small subset of tweets (like "panic") are distinctive and thus important. TF-IDF is a metric that normalizes the term frequency by dividing it by a measure of document frequency. It is computed as $\frac{ttff}{\log(ddff+1)}$ where $tttt$ is the number of times a particular term appears in a particular document and $ddtt$ is the number of documents in the dataset that have that term. When you set output_mode="tf_idf" in layer_text_vectorization it fills the document-terms matrix with tf-idf weights instead of zeros and ones. This way words such as "covid" and "supermarket" that appear in a large fraction of tweets get a lower weight.
- text_vectorizer %>% adapt(covid_train) fits the text_vectorizer to the training data, that is, it uses covid_train dataset to create vocabulary.
- covid_train_dtm= text_vectorizer(covid_train$OriginalTweet), creates the term document-matrix for tweets in training data and you can call text_vectorizer similarly on the test and validation data to create their document-term matrices.
- Now you are ready to train and evaluate a neural network on these document-term matrices.

Training, Tuning, and Evaluating a Neural Network Model

Q1. (5 pts) Create an ANN model with two hidden layers to classify tweets into three classes (“Negative”, “Neutral”, and “Positive”). Note: This is a multi-class classification problem so make sure that you are using a correct loss function as well correct number of neurons/units in the final/output layer with correct activation function.

Unlike the fashion Mnist dataset where each image had to be flattened into a one dimensional vector, for this dataset, you do not need the flatten layer before the first dense layer as each observation (tweet) is represented by its tf_idf weights and is one dimensional.

Once the model has completed its training, get the prediction for the test data as follows.:

```
predicted_labels==as.numeric(model %>% predict(covid_test_dtm) %>%k_argmax())
```

Note: The above code replaces `predict_classes` function used in fashion_mnist example in the lectures. It was brought to my attention that `predict_classes` is deprecated.

Use a cross table to compare these predicted labels to the true labels in the test data and interpret the table

Q2. (5 pts) Use “tfruns” package to tune your ANN’s hyper- parameters including the number of nodes in each hidden layer, the batch_size, and learning_rate). Validate each model on the validation set. Answer the following questions:

1- (2pt) Which model (which hyper-parameter combination) resulted in the best accuracy on the validation data? **Make sure that you print the returned value from tfruns and report the run with the highest validation accuracy. Note: the best run is not necessarily the first run. Print the entire table returned by tfruns to see which run has the highest validation accuracy and that would be your best run.**
2- (2pt) take a screenshot of the learning curves of your best model and save it. Does your best model overfit?
3- (1pt) Does your validation_loss stop decreasing after several epochs? If so, at roughly which epoch does your validation_loss stop decreasing?

Q3. (5 pts) Now that we tuned the hyperparameters and selected the best model, we don’t need to withhold validation data anymore and can use it for training. Add the validation data to the train data. You can do this by first, converting your covid_train_dtm and covid_val_dtm into matrices and then combining them using “rbind”. Make sure that you also combine the train and validation labels (sentiments). Now re-train your best model on this new training data and evaluate it on the test data. **Compute precision and recall for the positive/neutral/ and negative classes as you did for assignment 2.** How does this model perform compare to your naïve Bayes model in assignment 2?

Problem2—Predicting Bike Sharing Demand

For this problem, you will be working with the bike sharing demand dataset from Kaggle: https://www.kaggle.com/competitions/bike-sharing-demand/data?select=train.csv . The dataset is comprised of hourly bike rental data spanning two years (2011-2012). The goal is to predict the total count of bike rentals based on features such as date, temperature, whether it is holiday, working day etc. This data is a time-series data because the observations are in a sequence and there is a temporal order between them. However for the sake of this assignment, let’s assume that the observations are independent and identically distributed (i.i.d). click on the link above and read the data description on Kaggle. Then download bike-data.csv file from canvas

1. (2p) explore the overall structure of the dataset using the str() function. Get a summary statistics of each variable. Answer the following questions:
    - How many observations do you have in the data?
    - What is the type of each variable? Categorical (nominal or ordinal) or continuous?
    - Is there any missing value?
    - Draw the histogram of count. Interpret what you see in the histogram.

Feature Engineering

2. Remove the “registered” and “casual” variables. These are the count of registered and casual users and together they can perfectly predict “count” so we are removing them from the model and predict count from the other features.
3. (1p) The count variable is severely right-skewed. A skewed target variable can make a machine learning model biased. For instance, in this case lower counts are more frequent in the training data compared to higher counts . Therefore, a machine learning model trained on this data is less likely to successfully predict higher counts. There are different ways we can transform a right-skewed variable to a more bell-shape distribution. Common transformations for a right-skewed data includes log, square-root and cube-root transformations. We are going to use square root transformation here to make the distribution of count more bell-shaped. Set the count variable in your dataframe equal to square root of count and plot its distribution again.

4. (2 pt) Variable datetime is not useful in its current form. Convert this variable to “day of month”, “year”, “day of week”, ”month” and “hour” variables. You can use `as.POSIXlt` function to extract those features from datetime. Please see this reference for an example. Remove the original datetime variable after conversion.

5. (3pt) Variables “month”, “day of week”, “hour”, and “season” are categorical but they are also circular. This means these variables are periodic in nature. If we represent a circular variable like “month” with numeric indices 0-11 we are implying that the distance between month 10 (November) and month 11 (December) is much lower than the distance between month 11(December) and month 0 (January) which is not correct. At the other hand if we one-hot-encode the “month” variable we are ignoring the chronological ordering between month values and assume that the distance between every two months is equal. A better way to represent these variables is to map each value into a point in a circle where the lowest value appears next to the largest value in the circle. For instance, we can transform the “month” variable by creating “x” and “y” coordinates of the point in such circle using sin and cosine transformations as follows:

$$x_{month} = \cos\left(\frac{2\pi * month}{\max(month)}\right)$$
$$y_{month} = \sin\left(\frac{2\pi * month}{\max(month)}\right)$$

For instance, month 10 is converted to $x_{month} = \cos\left(\frac{2\pi*10}{11}\right)$ $and\ y_{month} = \sin\left(\frac{2\pi*10}{11}\right)$

Convert variables “month”, “day of week”, “hour”, and “season” to their x and y coordinates using sin and cosine transformation as explained above. Make sure to remove the original “month”, “day of week”, “hour”, and “season” variables after transformation.
Note: The “day of month” variable is also technically circular but this dataset only contains days 1-19 therefore we can just convert “day” variable to numeric indices using “as.numeric” function.

6. (2pt) Neural networks do not accept categorical variables and we must encode the categorical variables before training the network. One-hot-encode all the categorical variables in your dataset. Note: binary variables such as “holiday” and “workingday” are already converted to 0-1 and don’t need to be one-hot-encoded

7. Use set.seed(1) to set the random seed so I can reproduce your results.

8. Use Caret’s “createDataPartition” method as follows to partition the dataset into bikes_train, and bikes_test (use 90% for training and 10% for testing)

```
inTrain = createDataPartition(bikes$count, p=0.9, list=FALSE)
bikes_train = bikes[inTrain,]
bikes_test = bikes[-inTrain,]
```

where “bikes” is the name of your pre-processed data frame. The first line creates a random 90%-10% split of data such that the distribution of the target variable bikes$count is preserved in each split. The `list = FALSE`option avoids returning the data as a list. Instead, inTrain is a vector of indices used to get the training and test data.

9. (1pt) Set.seed(1) and further divide the bikes_train data into 90% training and 10% validation using Caret's "CreateDataPartition" function.

10. (2 pt) Scale the numeric attributes in the training data (except for the outcome variable, "count"). Use the column means and column standard deviations from the training data to scale both the validation and test data (please refer to slide 81, lecture 9). Note: You **should NOT scale the dummy variables you created in step 6.**

11. (5 pt) Create an ANN model to predict count from other attributes. Use at least two hidden layers. Use tfruns to tune your model's hyper-parameters including, the number of nodes in each hidden layer, the activation function in each hidden layer, batch_size, learning_rate, and the number of epochs). Validate each model on the validation set. Answer the following questions:

- Print the returned value from tf_runs to see the metrics for each run. Which run (which hyper-parameter combination) gave the best mean squared error on the validation data?
- Print the learning curve for your best model. Does your best model still overfit?
- Does your validation_loss stop decreasing after several epochs? If so, at roughly which epoch does your validation_loss stop decreasing?

Note: The "fit" function in keras does not accept a dataframe and only takes a matrix. If you want to pass a dataframe as training or validation data to the fit function, you must first use **as.matrix** function to convert it to matrix before passing it to the fit function; for example, **as.matrix(your_training_dataframe) or as.matrix(your_validation_dataframe)**

12. (5 pt) Measure the performance of your best model (after tuning) on the test set and compute its RMSE. **Note that you must reverse the square root transformation by taking the square of the predictions returned by the neural network model and compare it to the original count value (without square root transformation).** Doing this, helps us get the RMSE in the original scale.

13. (5 pt) Use a simple (or step wise) linear regression model to predict the count. Train and test your model on the same data you used to train and test your best neural network model. Compare the RMSE of the linear model on the test data with the RMSE of the neural network model. How does your neural network model compare to a simple linear model?

What to Turn in:
You need to create an Rnotebooke consisting of your code and answers to the questions outlined above for problems 1 and 2.

Format of the submission

The submission must be in two formats:

1- **A .html file which contains the preview of your notebook. When you click on preview in R an html file is created in the same directory as your notebook.**
2- **An .rmd file of your notebook**
3- **Any additional R script you used for creating your neural network models.**
4- **The learning curve from your best runs. You can just take a screenshot of your best model and submit it with the rest of your files.**