Lesson: Artificial Intelligence II

Fall Semester 2021

Student's name: Christina Christodoulou

I.D Number: LT1200027

Field of Studies: Language Technology

Homework 2

REPORT

In this assignment, two multi-class vaccine sentiment classifiers are developed in Python language using feed-forward neural networks. My solution was implemented using the Pytorch library. I ran the notebook in Google Colab using its GPU. The training and validation datasets were provided in csv form containing tweets and the labels 0 for neutral, 1 for anti-vax and 2 for pro-vax. Both models were implemented with pre-trained word embedding vectors from Twitter using GloVe, which I found here: https://github.com/stanfordnlp/GloVe. I downloaded the *glove.twitter.27B.zip* and used the text with the 25 dimension vectors (*glove.twitter.27B.25d*). For the development of the models, I experimented with the number of hidden layers and the number of their units, the activation functions, the loss function, the optimizer, the learning rate, the batch size and some regularization parameters, such as dropout and batch normalization. I plot the loss vs epochs and accuracy vs epochs as well as the ROC curve for both models to check the results. In the following paragraphs, I will explain in detail the steps followed in order to develop the two feed-forward neural network models in addition to my observations from the best model's plots. What is more, I compare the best model in this assignment with the best model in the previous assignment using softmax regression.

A. Explanation of script - Steps

First of all, the necessary libraries like torch, tqdm, nltk, sklearn, numpy, pandas and matplotlib were imported as well as the necessary packages ('stopwords' and 'punkt') were downloaded from nltk. Then, the English set of stopwords was set to use. To run this notebook, the following files were uploaded: *vaccine_train_set.csv, vaccine_validation_set.csv, glove_dict25d.pkl, glove.twitter.27B.25d.txt*. The colab notebook contains twelve functions and two classes that are combined and run one inside the other.

The first function is called *read_explore_dataframe(csv_file)*, which opens and reads a dataframe in csv form given as parameter using utf-8 encoding, as well as gets the values and number of values of the dataframe. It also checks whether there are empty values and duplicates and fills the empty values.

The second function is called *text_preprocessing(text)*, which takes as parameter a text and applies some pre-processing steps like removing unusual characters, urls, punctuation, emoticons, numbers and applying lowercasing.

The third function is called *get_columns(dataframe, feature, label)*, which takes as parameters a dataframe, the name of the feature's column and the name of the label's column. More specifically, it selects the columns of interest and puts them in a new dataframe. It checks the distribution of each class and then separates the values of the label's column into 3 classes, 0 for neutral, 1 for antivax and 2 for provax. Since I noticed an imbalance data problem mainly in class 1, I decided to apply resampling using the *resample* function offered by sklearn. Thus, I created a new dataframe with only the 3 classes and used it as the number of samples in the function. I set the replace parameter as True, so it means that the same samples can be used multiple times. At first, I simply tried upsampling class 1 using the number of samples from the other two classes, but class 1 ended up becoming the majority class and the other two classes the minority classes, which posed once again an imbalance problem. For this reason, I decided to create the same number of values for all the classes in order to achieve data balance and fair predictions of the model for each class. What is more, the previous function is applied to the text (feature) column for pre-processing. After pre-processing, the feature column is assigned as the x input value and the label column as the y target value, which are returned by the function.

The fourth function is called *create_tensors(x, y)*. It takes as inputs the x and y values given from the previous function and turns them into tensors using the *torch.tensor* and *torch.squeeze* functions. I converted them to torch.float type, as I encountered many errors till I found the solution to fix the issue.

In the next kernel, I first open the glove file in txt form and get the words and their corresponding vectors and put them into separate lists. Then, I save the word and its vectors into pickle form to load and use later. The fifth function contained in this kernel is called *find_words_in_glove(dataset)*, which takes as parameter the x input value of the train dataset. It checks whether a word from the train dataset exists in the glove dictionary with 25 vector dimension and loads its pre-trained word vector. Otherwise, it initializes the vector with zeros. It also finds the mean vector for each word. It returns the mean vectors in array, the words that were found in glove and the length of the matrix.

The sixth function is called *train_neural_network_model(epochs, model, dataloader, optimizer, loss_func)*, which takes as parameters the number of epochs, the initialized neural network model, the dataset split into batches as well as the initialized optimizer and loss function. It trains a feed-forward neural network model step by step. More specifically, it performs forward propagation to get the output values, then calculates the loss, clears up the accumulated gradients, performs backward

propagation and updates the parameters. Then, it calculates predictions, the loss and accuracy for each epoch. It shows the training loss and training accuracy for each epoch.

The seventh function is called *evaluate_neural_network_model(epochs, model, dataloader ,optimizer, loss_func),* which takes as parameters the number of epochs, the initialized neural network model, the dataset split into batches as well as the initialized optimizer and loss function. It evaluates a feed-forward neural network model. More particularly, it performs forward propagation to get the output values and calculates the loss. Then, it calculates predictions, the loss and accuracy for each epoch. It shows the validation loss and validation accuracy for each epoch. I added the *tqdm* function for the progress bar to appear during the training and evaluation process.

The eight function is called *calculate_metrics(y_true, y_pred)*, which takes as parameters the y true values and the x predicted values. It calculates all the metrics asked for this assignment, namely the precision, F1 score and recall. It also calculates the accuracy of the feed-forward neural network model and prints the classification matrix as a general summary of all the metrics.

Next, is a class named *feedforward_neural_network_model(nn.Module)*, which initializes the attributes of the first feed-forward neural network model. It contains 2 hidden layers, the RELU activation function applied after each layer as well as regularization parameters like dropout of 0.3 and batch normalization before the output layer. It contains a *forward(self,x)* function, which performs the forward propagation.

The main function running for the first model including all the rest functions is called *data_train_first_model(csv_file_train, csv_file_test, glove_file_pickle, feature, label)*, which takes as parameters the train and the validation/test dataset, the glove dictionary in pickle, the feature column and the label column. It uses the first functions to prepare the train and the validation/test dataset by reading the csv files, checking for duplicates and nan values as well as getting the necessary columns, apply resampling and getting the x and y values. It then loads the glove file from pickle and uses it with the function to find how many words in the x train dataset exist in the pre-trained glove dictionary. It converts the train and validation sets into tensors and splits them into batches using TensorDataset and DataLoader. I selected batch size 6 and shuffled only the train set to avoid bias. Apart from this, the function defines the dimensions of each layer. The input layer is the size of the x train tensor and the output layer is the same size as the number of classes, namely 3. The first hidden layer's size is 32 and the second's is 16. Next, the feed-forward neural network is defined using the first class defined in this colab notebook as well as the Cross entropy loss function, which is appropriate for multiclass classification and the Adam optimizer with learning rate set as 0.0001 and weight_decay set as 1e-4. Furthermore, four empty lists are initialized to store the training accuracy, training loss, validation accuracy and validation loss respectively. Then comes the train and test phase, where the function iterates over the number of epochs (here set as 30), trains and evaluates the model and fills

the empty lists. The filled lists are then used to plot the loss vs epochs and the accuracy vs epochs. The main function also makes predictions using the test set and shows the calculated metrics. Finally, the ROC curve plot appears showing the curve for each class.

What is more, a class named *Feed_forward_Neural_Net_Model(nn.Module*), which initializes the attributes of the second feed-forward neural network model. This class constructs a deeper neural network network than the first model as it contains 4 hidden layers. It contains the RELU activation function applied after each layer except from the last hidden layer where the Softmax activation function is applied, as well as regularization parameters, like dropout of 0.3 in the first and last hidden layer and batch normalization before the output layer. It contains a *forward(self,x)* function, which performs the forward propagation.

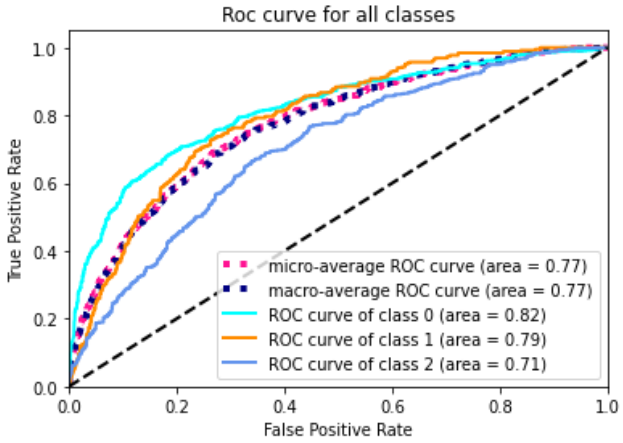
Finally, the main function running for the second model including all the rest functions is called *data_train_second_model(csv_file_train, csv_file_test, glove_file_pickle, feature, label)*, which takes as parameters the train and the validation/test dataset, the glove dictionary in pickle, the feature column and the label column. It uses the first functions to prepare the train and the validation/test dataset by reading the csv files, checking for duplicates and nan values as well as getting the necessary columns, apply resampling and getting the x and y values. It then loads the glove file from pickle and uses it with the function to find how many words in the x train dataset exist in the pre-trained glove dictionary. It converts the train and validation sets into tensors and splits them into batches using TensorDataset and DataLoader. I selected batch size 10 and shuffled only the train set to avoid bias. Apart from this, the function defines the dimensions of each layer. The input layer is the size of the x train tensor and the output layer is the same size as the number of classes, namely 3. The first hidden layer's size is 500, the second's is 256, the third's is 128 and the fourth's is 64. Next, the feed-forward neural network is defined using the second class defined in this colab notebook as well as the Cross entropy loss function, which is appropriate for multiclass classification and the SGD optimizer with learning rate set as 0.01 and weight_decay set as 1e-4. Furthermore, four empty lists are initialized to store the training accuracy, training loss, validation accuracy and validation loss respectively. The train and test phase follow, where the function iterates over the number of epochs (here set as 40), trains and evaluates the model and fills the empty lists. Since the first model needed at least 20 epochs to reach the minimum loss, I figured that this model would need larger number of epochs due to its complexity. The filled lists are then used to plot the loss vs epochs and the accuracy vs epochs. The main function also makes predictions using the test set and shows the calculated metrics. Finally, the ROC curve plot appears showing the curve for each class.

I experimented a lot with the number of hidden layers and the number of their units. I selected to construct one simpler and one deeper, more complex neural network to compare their results. I used only the RELU and the Softmax activation functions. I found difficulty using another loss function

expect from the Cross-entropy, despite experimenting with others like MultiLabelSoftMarginLoss and BCE Loss and MSE Loss as I encountered many errors and was not able to fix them. I experimented with two optimizers too, the Adam and the SGD optimizer by trying them with different learning rates (0.01, 0.001, 0.0001, 0.00001). I also changed the batch size many times (128,200,64,32,10,6,4) to observe the result. I ended up selecting the aforementioned dimensions and parameters due to overfitting issues that arose during experimenting. For this reason, I added the dropout and the batch normalization regularization techniques. From the learning curves, I noticed that there was a huge gap between the train and the validation line both in loss and accuracy. The loss and accuracy lines were fluctuating over the training examples.

B. PLOTS AND ROC CURVE (FIRST MODEL)

It seems that the model is overfitting, since the training loss continues to decrease with experience and the validation loss fluctuates over the training examples, as it decreases to some points and begins increasing again. [1]I experimented with various number of epochs combined with different parameters and observed that the overfitting issue remains. The same issue applies to the second model as well.



[1] https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

The ROC curve provides us information about the model's ability to distinguish between classes, since it is plotted using the True Positive Values on the y axis and the False Positive Values on the x axis. The code of creating this ROC curve for multi-class classification was found in the site of sklearn (https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html). It shows that the model has 82% chance of distinguishing class 0, 79% chance of distinguishing class 1 and 71% chance of distinguishing class 2. These are considered good scores for the efficiency of this model.

C. RESULTS - COMPARISON

From the board below, it appears that both models did not offer satisfying results perhaps due to the low dimensions used to train the models (25). Their results do not vary. The precision score seems more distributed among the three classes, whereas the recall score in the third class is much lower than in the rest classes. The F1-score is also very low in the third class because of the low recall, since it is calculated by combining the precision and the recall. However, the first model with the simpler feed-forward neural network achieved slightly higher accuracy score than the first with minimum cost 92.5, while the second model achieved minimum cost approximately 93.0. The minimum cost is very high and it shows the models did not go as well as expected.

FFNNM Models	Accuracy	Precision (class 0,1,2)	Recall (class 0,1,2)	F-measure (class 0,1,2)
First FFNNM 50 dim	58.10%	57.5%, 59.0%, 57.7%	73.2%, 69.4%, 31.9%	64.4%, 63.7%, 41.1%
Second FFNNM 50 dim	57.30%	61.8%, 55.4%, 52.8%	68.9%, 70.9%, 31.9%	0.65230004 0.622598 0.39814309

Although the two feed-forward neural network models presented similar results, I selected the first model as the best in order to compare it with the best softmax regression model from the previous assignment. It is evident that the softmax regression model is the best model so far, since it achieved a much higher score in all the metrics compared to the feed-forward neural network. This can be mainly due to the fact that the latter was trained with a larger number of features using CountVectorizer, whereas the former was trained with very small dimensions of pre-trained word embedding vectors as input.

Models	Accuracy	Precision (class 0,1,2)	Recall (class 0,1,2)	F-measure (class 0,1,2)
Second Feed-Forward Neural Network Model 25 dim	58.10%	57.5%, 59.0%, 57.7%	73.2%, 69.4%, 31.9%	64.4%, 63.7%, 41.1%
Softmax Regression Model CountVectorizer 21400 feat	92.48%	93.3%, 93.6%, 90.5%	88.5%, 97.8%, 90.7%	90.8%, 95.7%, 90.6%