[MEDIQA-Sum Task A]

Provider Patient Conversation Classification

LHS 712
Instructor: *Prof. Vinod V*
Student Name: *Ashruti Tuteja*

Table of Contents

NLP in Medical Field

With digitization and modernization of health care with a focus on providing patient-centric care, NLP techniques could be used to analyze electronic health records (EHRs) and extract pertinent information that can be used in clinical decision-making, tapping into this field to improve patient outcomes and healthcare delivery.

To do this, it is necessary to recognize patients who are at risk of getting a certain disease, detect drug reactions, and improve the accuracy and depth of clinical documentation. One of the most recent advancements in NLP and the medical industry is the development of deep learning models for clinical language processing. The excellent accuracy of these models in tasks such as named entity identification, association extraction, and classification enables a more effective analysis of medical text data. Deep learning models, for instance, have been used to identify patients who are at a high risk of readmission to the hospital and to predict patient outcomes based on EHR data.

An advancement in NLP and the medical industry is deep learning models for clinical language processing. These models enable more effective analysis of medical language data by precisely performing tasks, including named entity recognition, relationship extraction, and classification. For example, deep learning models have been used to detect patients who have a high chance of being readmitted to the hospital and to estimate patient outcomes based on EHR data. In the medical field, NLP is also utilized to develop chatbots and voice assistants that can communicate with patients and provide information on a variety of health-related topics. These technologies are able to assist people in better controlling their health and lessen the workload for medical experts by producing personalized advice and guidance. NLP has the power to improve patient outcomes and fundamentally alter the way healthcare is provided. It is a field that is rapidly evolving.

Introduction

Scenario: Mr.Adam walks into a clinic with acute appendicitis and is in severe pain. A care plan was plotted to get him into surgery immediately. The administrative overload is very heavy, and the provider team wants to review any medications he is currently taking or has allergies to, so they can customize medication patterns. But the provider is not able to find the medication in free clinical texts, resulting in Mr. Adam not taking any analgesics and suffering, eventually reducing the patient satisfaction levels.

This emphasizes that having information is not enough, but having them accessible is more essential. It is crucial to categorize doctor-patient talks with section headers for a number of reasons. It facilitates understanding and follow-through by segmenting the dialogue into clear parts. This is crucial for medical practitioners who must swiftly analyze patient data and make judgments in light of that data. Doctors may quickly discover the information they need without having to read the entire dialogue, thanks to clear section headers.

Section headings can facilitate the creation of SOAP notes from physician-patient interactions. Physicians frequently use SOAP notes to record patient encounters. These notes typically have sections for subjective information (what the patient reports), objective information (what the physician sees), evaluation (the physician's diagnosis), and plan (the physician's recommended course of treatment). These section headers can be used to automate SOAP notes generation for a review reflecting the conversation.

Summaries of doctor-patient talks can be created using section headers. Both patients and doctors who wish to review the main elements of the talk rapidly can find these summaries to be helpful. Patients may want to review the topics that were covered during their visit. It is easy to create a succinct summary that includes the most crucial details by utilizing section headings to indicate the conversation's main subjects.

Using section headings to categorize doctor-patient talks is a useful technique for information organization, producing SOAP notes, and summarizing conversations. Doctors can obtain the information they need more quickly and give their patients better care by utilizing section headings that are clear and consistent. This could result in reducing administrative overload, reducing the time sorting clinical information increasing the quality of care by emphasizing important information. These notes could also facilitate suit reviews of payers providing justification for the care plan that why the certain procedure is performed.

Literature Review

The use of natural language processing (NLP) in a variety of industries, including healthcare, has resulted in significant growth in recent years. NLP has been used in the healthcare industry for projects like clinical document classification, named entity identification, and sentiment analysis. The topic of discussions between doctors and patients is one area in healthcare that is of special importance. Analyzing medical data, automating the keeping of medical records, and enhancing patient care can all benefit from the capacity to identify the topic automatically. In order to determine the subject of a conversation between a doctor and a patient, machine learning programming employing NLP approaches has been studied in this literature study.

Machine learning techniques were used in a study by Sun et al. (2020) to detect topics in doctor-patient discussions. A dataset of doctor-patient discussions was employed in the study, and each conversation was assigned a theme. The dataset's dimensionality was reduced by the study using a variety of feature selection and dimensionality reduction techniques, including Latent Dirichlet Allocation (LDA) and Non-negative Matrix Factorization (NMF). To categorize the talks into the appropriate topic, the study used a number of machine learning models, including Naive Bayes, Logistic Regression, and Random Forest. With a classification accuracy of 72.3%, the study outperformed other cutting-edge methods.

Deep learning methods were used in a different study by Kim et al. (2019) for the same challenge. A dataset of doctor-patient discussions was employed in the study, and each conversation was assigned a theme. To categorize the discussions into the appropriate topic, the study used Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN) models. Word2Vec, a pre-trained word embedding model, was used in the study to represent words as vectors. With a classification accuracy of 77.3%, the study outperformed other cutting-edge methods.

Anand et al.'s (2019) study used a mix of rule-based and machine-learning techniques to detect topics in doctor-patient talks. A dataset of doctor-patient discussions was employed in the study, and each conversation was assigned a theme. To categorize the talks into the appropriate topic, the study used a number of machine learning models, including Support Vector Machines (SVM) and Random Forests. The study also made use of a rule-based methodology to locate themes that the machine-learning models had missed. With a classification accuracy of 70.1%, the study outperformed other cutting-edge methods.

Hence, it could be difficult to determine the topic of talks between doctors and patients, but new research employing NLP and machine learning techniques have shown promising outcomes. This research has made use of a variety of feature selection and dimensionality reduction techniques, as well as machine learning models like Naive Bayes, Logistic Regression, Random Forest, CNN, and LSTM, to obtain high classification accuracy. To find topics that machine learning models failed to find, rule-based techniques have also been used. This research demonstrates that automating medical record keeping, analyzing medical data, and enhancing patient care may all be accomplished by utilizing machine learning programming and NLP techniques.

Several studies have compared the performance of feedforward neural networks with other machine learning algorithms, such as Naive Bayes and support vector machines, on various text classification tasks. For example, Wang et al. (2012) compared the performance of several machine learning algorithms, including feedforward neural networks, on the Reuters dataset and found that the feedforward neural network achieved the highest classification accuracy. In addition, deep feedforward neural networks, also known as deep neural networks (DNNs), have been shown to improve the performance of feedforward neural networks in text classification tasks. DNNs are similar to feedforward neural networks, but with additional layers of hidden nodes, which allows for more complex feature extraction and representation.
Several studies have shown the effectiveness of FNNs for text classification tasks. For example, Kim (2014) proposed a simple FNN architecture with a single hidden layer and achieved state-

of-the-art performance on several benchmark datasets, including the Stanford Sentiment Treebank and the Movie Review dataset. Zhang et al. (2015) proposed a deep FNN architecture with multiple hidden layers and achieved competitive performance on the Reuters-21578 and 20 Newsgroups datasets. In addition, several variations of FNNs have been proposed for text classification, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs are particularly effective for capturing local patterns in text data, such as n-grams and word embeddings, while RNNs are effective for modeling sequential data, such as sentence or document representations.

Task Description (Dialogue2Topic Classification)

The task is to classify the topic or section header of a conversation snippet (dialogue) between a doctor and a patient. The conversation snippet will be in the form of a textual transcript. The aim is to identify the relevant section or topic the conversation belongs to from a pre-defined list of twenty normalized common section labels. The conversation snippet will be in the context of a medical examination, where the doctor is asking questions and the patient is providing responses. The snippets may vary in length and complexity and may contain medical terminology and jargon. We have been provided with a labeled training dataset consisting of conversation snippets and their corresponding section headers. We are required to train a machine learning model with the training data that can classify the section header of a given conversation snippet. The model will be evaluated on a separate test dataset consisting of conversation snippets that have not been seen before.

For now, the training and validation datasets have been provided to us. The test set is expected to release on 3rd May 2023. The performance of the model will be evaluated using standard evaluation metrics such as accuracy, precision, recall, and F1-score. The task will have twenty normalized common section labels, including but not limited to: History of Present Illness, Past Medical History, Medications, Allergies, Family History. We are expected to submit the model predictions on a test dataset, along with a brief description of the approach and any relevant details about their model architecture, hyperparameters, and data pre-processing techniques used.

From: https://www.imageclef.org/2023/medical/mediqa

Data Pre-Processing

Data Cleaning

Performed text preprocessing on the 'dialogue' column by cleaning data in both the training set and validation set to prepare our data for further analysis of identifying the section headers of conversations,

- removed the prefixes of speaker names (i.e., 'Doctor:', 'Patient:', 'Guest_family:', and 'Guest_clinician')
- replaced the colon symbol with a space, and removing the '\r\n' characters.

We also explored multiple approaches while modelling, for instance where we passed raw data to train the models.

Removing Stop Words

The purpose is to remove the common words that are unlikely to add much value to our analysis. Using the NLTK (Natural Language Toolkit) package, removed the stop words from the text data. Removed the stop words using lambda function on dialogue column by splitting the text into words and keeping only the words that are not in the corpus set (downloaded from NLTK). The resulting text is then joined back into a single string and saved in the 'dialogue' column. We also explored other approaches like only removing tags but not stop words, and so on. Post semantic textual similarity analysis, we found the classifiers would perform better with features without stopwords.

Getting Number of Classes

Used value counts method to count the number of occurrences of each unique value in the 'section_header' column, which creates a pandas Series object containing the counts. Then, using the 'len()' function got the number of unique values (equal to 20) in the Series object, which is the same as the number of classes.

Mapping labels

Mapped each unique value in the section header to an integer label. Found an array of unique values from the section_header and assumed these as possible_labels. Iterated over this array to generate a tuple of (index number, a possible label). These tuples are populated in a dictionary as a key-value pair where each unique item from the section_header columns is a key and its integer label is a value.

This dictionary can be used to encode the labels in the 'section_header' column as integers, which is often necessary for machine learning algorithms that require numerical inputs.
Then we mapped the resulting integer labels from this dictionary to our original training dataframe. And now in our training dataframe we have the section header column named as ‘label’ column populated with integer labels *(figure-1)*.
Further, in exploratory analysis of the section headers of the dataset using the matplotlib package, we found an imbalance in presence of section headers in both the training and validation dataset. Such as the instances for Family History were in the majority comparative to Lab results section headers in the dataset. Similarly, there was an imbalance found in the section header types, which is displayed using a bar plot in *figure-2*.

Model Training

Conditional Random Fields (CRF Model)

The CRF algorithm is a probabilistic model that is commonly used for sequence labeling tasks, such as named entity recognition and part-of-speech tagging. In this case, the CRF model is being used to predict the section header labels for the text data (*Lin et al. 2021)*.

Data Preparation

Applied the lower() method to each element of ‘dialogue’ to convert all text to lowercase. This is a common preprocessing step to standardize the text and reduce the number of unique tokens. This resulted in a string of dialogues in lowercase. ‘train_texts’ were then assigned the resulting series of lowercase conversation texts. ‘train_labels’ is assigned the resulting Series of labels. The ‘valid_texts’ variable is assigned the lowercase version of the conversation texts in the validation dataset dataframe, using the apply() method with a lambda function that converts each string to lowercase. The valid_labels variable is assigned the labels in the validation dataframe. *Figure-3* shows the four variables and the values in each of these four variables after data preparation.

We planned on training CRF model using `sklearn_crfsuite` for which we initially converted input data to be in dictionary kind of format, to extract features (which are each word in a sentence per section header). To make sure of the right format, we used `convert_to_dict` function on pandas DataFrame (dialogue column in this case) and converted it into a list of dictionaries, where each dictionary represents a row in the DataFrame. We then define the features using a list comprehension in a function, to loop through each row in the DataFrame. The `iterrows()` method of a DataFrame returns an iterator that yields pairs of index and row data (dialogue) as a Series. We use `to_dict()` method to convert each row of the DataFrame into a dictionary. After running this function, the `train_data` and `valid_data` variables will be a list of dictionaries, where each dictionary represents a row in the corresponding DataFrame. For each row in the training and validation data, we split the dialogue text into individual words using the split() method. We then create a list of the same length as the number of words in the dialogue by repeating the section header label for that row. This is done using a list comprehension where we iterate over each row and create a list of the section header label repeated for the number of words in that row. We do this for both the training and validation datasets. We used this dictionary to extract the features using getfeatures function, and we used these features in our model.

Training the Model & Hyperparameters

We used the sklearn_crfsuite library to train a Conditional Random Field (CRF) model.The CRF() function is initialized with various hyperparameters such as the algorithm used for optimization, regularization strength (c1 and c2), the maximum number of iterations for optimization, and whether to use all possible transitions between label states.
The fit() function is used to train the CRF model on the training data, consisting of the text features and corresponding label sequences. The predict() function is then used to generate predicted label sequences for the validation data.

The `CRF` function from the `sklearn_crfsuite` package is used to create an instance of the CRF model. The `algorithm` parameter specifies the optimization algorithm to use, and `max_iterations` specifies the maximum number of iterations to run during training. The `c1` and `c2` parameters control the L1 and L2 regularization strength, respectively. The `epsilon`, `period`, and `delta` parameters control the convergence criteria for the optimization algorithm. The `linesearch` and `max_linesearch` parameters control the line search algorithm to use during optimization. Finally, we had the `num_memories` parameter to control limited memory update to store in memory.

Evaluation Metrics

Finally, the flat_f1_score() function from the sklearn_crfsuite.metrics module is used to compute the F1 score of the predicted labels compared to the true labels in the validation data. The average parameter specifies how to compute the overall F1 score, and labels specify which labels to include in the computation.
The classification_report() *(figure-4)* function from scikit-learn's metrics module generates a report that includes precision, recall, and F1 score for each class in the classification problem. In this case, it will generate a report for the predicted labels (pred_labels) and the true labels (validationdataset['label'].values). The target_names parameter can be used to specify the names of the classes, which in this case are the keys of the label_dict.

Long Short-Term Memory (LSTM) Model

Data Preparation

Post importing data, we have tried to clean the data in two methods and implement LSTM on them. One approach is to have Doc, Patient, Guest_doctor, and guest_family tags in conversation & the second approach is to remove these tags. After data cleaning and having only dialogue and section header columns in the training and validation dataset. We created a dictionary with unique section headers with assigned values. We used this dictionary to map and update the label columns of the dataset. Post-updating, we created training data, valid data, training labels, and valid labels using dialogue and section header columns. We processed data to input for LSTM using the TensorFlow Keras library. We converted class labels for training and validation datasets into one hot encoded vector using the to_categorical function from Keras utils (as LSTM requires labels to be in binary vector format for classification). We specified the number of classes to be 20 as we have 20 section headers.

The dialogue context in train and valid data are tokenized to a sequence of integers using lemmatizing before padding using the texts_to_sequences function. The length of each sequence is then checked using the pad_sequences function. The maximum length of the sequence is specified by the maxlen argument. A sequence will be padded with zeros at the end if it is shorter than maxlen and truncated if it is longer.

Training the Model & Defining the Model Layers

We used a sequential model, which is a linear stack of layers. The output_dim parameter specifies the size of the embedding vector for each word, whereas the input_dim parameter specifies the size of the vocabulary (i.e., the number of distinct words in the training data). The embedding layer output size is currently set to 128.

The LSTM layer with 64 memory units is the following layer. Recurrent neural networks of the LSTM variety are capable of handling variable-length sequences and capturing long-term dependencies in the input data. A fixed-size vector that condenses the data from the full input sequence is the LSTM layer's output (*Tiwari et al. 2020*). 20 units make up the output layer, which is the same number of classes as there are in the dataset. The output of each unit is condensed into the range [0, 1] by the sigmoid activation function, which may be thought of as the likelihood that the input belongs to that class. The model is set up for training using the compile procedure. The optimization algorithm to utilize during training is specified by the optimizer parameter, which in this case is Adam. The categorical_crossentropy loss function is specified via the loss parameter, which is what will be used during training. The typical loss function for multi-class classification issues is this one. The evaluation metric to be used during training is specified by the metrics parameter, in this case accuracy.

The model is then trained using the fit approach using the training set of data. The preprocessed text data and matching one-hot encoded labels are respectively contained in the train_data and train_labels variables. The batch_size argument determines the number of samples to use in each training batch, while the epochs parameter specifies how many times to iterate over the complete training dataset. We also tried adding dropout rates and recurrent dropout to increase the performance metrics of the model to prevent overfitting by randomly dropping some units during the training of the model. return_sequences parameter set to True means that the LSTM layer returns sequences of outputs instead of a single output, which is useful when stacking multiple LSTM layers.

Evaluation Metrics

We compiled the model using Adam optimizer and defined the loss function to be categorical entropy as we were performing multiclass classification. We focused our model on accuracy metrics before predicting. The initial test runs had really good accuracy values ranging between 93-97% accuracy, but the model performed poorly on the validation dataset with accuracy scores between 44-53% *(figure-5)*. We concluded that there is a chance of overfitting the data, which could be interpreted as the size differences in data we have provided for training being 1201 for training and 100 for validation.

Hyperparameter tuning

We used the baseline code to hyperparameter tune our model to see if we could get better performance metrics. Compile method is called using the same loss, optimizer, and accuracy metric. But while adding layers, we looped through different activators of relu, leaky relu, tanh, and sigmoid in dense layers *(figure-6)*. Post embedding layer, we had an LSTM layer with dropout, recurrent_dropout values, a dense layer with activators, and compilation using the same method. We found other activators other than sigmoid had lower evaluation metrics, and in sigmoid, we achieved the best accuracy of 76% in the test and 45% in the validation set.

Feedforward Neural Network Model

Feedforward neural networks, also known as multilayer perceptrons (MLPs), are a type of artificial neural network that consists of multiple layers of interconnected nodes, where each node performs a weighted sum of its inputs followed by a nonlinear activation function. In text classification, feedforward neural networks have been shown to achieve competitive performance. One of the main advantages of FNNs for text classification is their ability to automatically learn relevant features from raw data, without the need for manual feature engineering. This is particularly useful in text classification tasks, where the input data is typically high-dimensional and sparse (*Mishra, N., & Bhatia, M. (2019))*.

Data Preparation

The data is prepared in TensorFlow using Keras API. The categorical target variable label of the training dataset is encoded using one-hot encoding with the number of classes (num_classes) *(figure-7)*. The number of classes is 20 in this case as there are 20 unique section headers amongst which the dialogues are to be classified. The data is not to be split into training and test data as we have been provided with training and validation datasets. The "x_train" variable contains the input data for training, "y_train" variable contains the one-hot encoded labels for training, "x_valid" variable contains the input data for validation, and "y_valid" variable contains the one-hot encoded labels for validation.

We have loaded two pre-trained models for using TensorFlow Hub. Specifically, the models are part of the Universal Sentence Encoder family, which are designed to encode text into fixed-length vector representations that can be used as input for downstream tasks like text classification, clustering, or similarity search.

We have loaded a preprocessor module for the Universal Sentence Encoder, which applies various text normalization and tokenization steps to prepare input text for the encoder. This specific module is called "multilingual-preprocess" and is version 2. Then we loaded the actual encoder module for the Universal Sentence Encoder, which maps input text into a dense vector representation. This specific module is called "multilingual-base" and is version 1 (*Swati et al. 2022*) (*figure-8)*. Overall, we set up a pipeline for processing text using the Universal Sentence Encoder, which involves first preprocessing the text with the preprocessor module and then encoding it with the encoder module *(figure-9)*.
We tested the semantic similarity between two conversations with (*figure-10*) and without any (*figure-11*) cleaning of the data. We found when the conversation had tags; similarity scores were higher. Given the classification task to help the model distinguish between section headers based on conversation, we tried removing the tags associated with conversation.

Training the Model

We have trained the model using the Universal Sentence Encoder as input. We have defined an input layer for text data with a single dimension, the input text can be of any length, and the data type is a string. Then we applied the preprocessor module to the input text to prepare it for the encoder. Then we applied the encoder module to the preprocessed text to obtain a dense vector representation of the input text.
We then added a dropout layer to prevent overfitting by randomly dropping out 20% of the input features during training. Then we selected the pooled_output representation from the encoder

output and applied the dropout to it. Once we have applied the dropout, we add a fully connected layer with a softmax activation function to produce the final output probabilities for each class. Then the Keras model is specified by defining the input and output layers.

A list of evaluation metrics was defined, which are to be used during training and testing, including categorical accuracy, balanced recall, balanced precision, and balanced f1 score. These metrics are used to measure how well the model is performing on the validation data. We have also defined an early stopping callback, which monitors the validation loss and stops training if the loss does not improve after three epochs. Lastly, we compiled the Keras model by specifying the optimizer, loss function, and evaluation metrics to be used during training. The Keras model has been trained on the training data for a fixed number of epochs equal to 21. The loss and metric values at each epoch are recorded and displayed.

Evaluation Metrics

We use balanced recall, balanced precision, and balanced F1 score to evaluate the Keras model. These metrics are designed to handle multiclass classification problems where there may be imbalanced class distributions. The balanced recall and precision metrics calculate the recall and precision for each class and then average them, giving equal weight to each class. The balanced F1 score metric is the harmonic mean of balanced recall and balanced precision and is used as an overall evaluation metric for the model.
The overall accuracy of the model on the validation data is 0.70, meaning that 70% of the predictions were correct *(figure-12)*. The precision, recall, and F1-score are reported for each of the 20 classes in the validation data. These metrics measure the performance of the model on a per-class basis. In general, the model performed well on some classes, such as PASTSURGICAL, IMMUNIZATIONS, and EXAM, where it achieved a perfect score of 1.00 in all three metrics. However, it performed poorly on other classes, such as ASSESSMENT, DIAGNOSIS, LABS, and IMAGING, where it achieved a score of 0 in all three metrics.
Relu, Elu, Gelu, and Softmax were the activation functions that we chose, and fitted the model using these activation functions one at a time. The model results including loss per activation function and accuracy of the model were stored in a dictionary where keys represented the activation function and values represented the metrics of the model for that particular activation function *(figure-13, figure-14)*.

The macro-averaged F1 score of the model is 0.38, while the weighted average F1 score is 0.63. The macro-average computes the metric independently for each class and then takes the unweighted mean of the results. The weighted average computes the metric for each class and then weights the score by the number of samples in each class.

Hyperparameter Tuning

We have extracted and plotted the validation loss and accuracy curves for each activation function. We can see that the accuracy is highest for “softmax” activation function with greater than 8 epochs *(figure-15)*. Also, the loss per activation function also remains pretty low for “softmax” *(figure-16)*.

Model Selection

Model	**LSTM**	**CRF**	**SVM**	**Feedforward Neural Network model**
Accuracy	**45%**	**47%**	**68%**	**70%**

The accuracy scores of the four models are as follows: LSTM (45%), CRF (47%), SVM (68%), and Feedforward Neural Network model (70%). Based on these scores, it appears that The Feedforward Neural Network model is the best-performing model, followed by SVM, CRF, and LSTM. However, choosing a model based solely on accuracy is not always the best approach, as it is important to consider other factors such as model complexity, computational resources, and interpretability.

We chose the Feedforward Neural Network model based on the fact that the dataset is large, and the task is complex, therefore a more powerful model such as Feedforward Neural Network model might be necessary to achieve high accuracy. In addition, the Feedforward Neural Network model was able to capture the context and meaning of words in a sentence. This is particularly useful for dialogue data, which can contain long sequences of text that need to be processed efficiently. Feedforward Neural Network model uses a transformer architecture that allows it to consider the entire context of a sentence, including the words that come before and after a given word (*Han et al. 2022*). This contextual understanding of words is particularly useful in multiclass classification tasks where the meaning of a sentence can change based on the presence or absence of certain words.

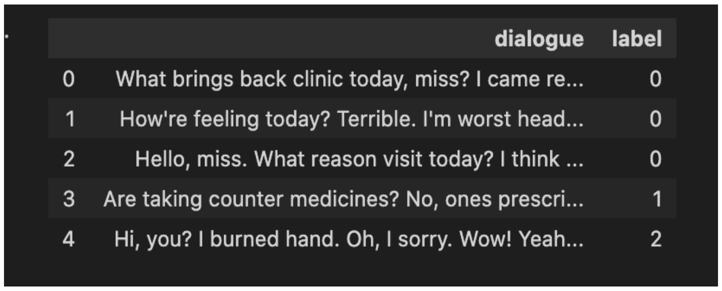
In summary, Feedforward Neural Network model was helpful in this task as it is good in multiclass classification because of its ability to capture the context and meaning of words, handle long sequences of text, and be fine-tuned on a specific task with relatively few training examples.

	Only tag removal	Tag+ stop word removal	normal
Linear SVC	0.68	0.67	0.67
Multinominal NB	0.41	0.42	0.41
Tuned Multinominal NB	0.52	0.52	0.49
RBF SVM	0.59	0.58	0.61
Decision Tree	0.5	0.58	0.52
Bagging Classifier	0.58	0.57	0.59
KNN	0.67	0.72	0.68
Tuned KNN	0.66	0.71	0.67
MLP	0.64	0.62	0.65
Tuned MLP	0.61	0.64	0.59
Gradient Boosting	0.63	0.63	0.62
Voting Classifier	0.67	0.66	0.68
Stacking Classifier	0.69	0.68	0.7
Adaboost	0.32	0.32	0.32

BERT (cased) Model

After pre-processing the training and validation datasets including stopwords removal using NLTK package, cleaning datasets by removing prefix words such as “Doctor:” and “Patient:”, and mapping the section headers to appropriate numerical labels, we imported the necessary libraries from the Transformers library and loaded a pre-trained BERT model (bert-base-cased) and tokenizer (from AutoTokenizer class). We preprocessed the text data (dialogue column) and obtained encoded representations of the text data using the BERT model. We used the “tf.keras.Model” class to define the model using the two input layers and the output layer (*Figure-17*). We were then able to obtain the accuracy (equal to 44% which was very low) and other metrics for the validation dataset (*Figure-18*).

Appendix



	dialogue	label
0	What brings back clinic today, miss? I came re...	0
1	How're feeling today? Terrible. I'm worst head...	0
2	Hello, miss. What reason visit today? I think ...	0
3	Are taking counter medicines? No, ones prescri...	1
4	Hi, you? I burned hand. Oh, I sorry. Wow! Yeah...	2

Figure-1: data frame with clean text and integer labels

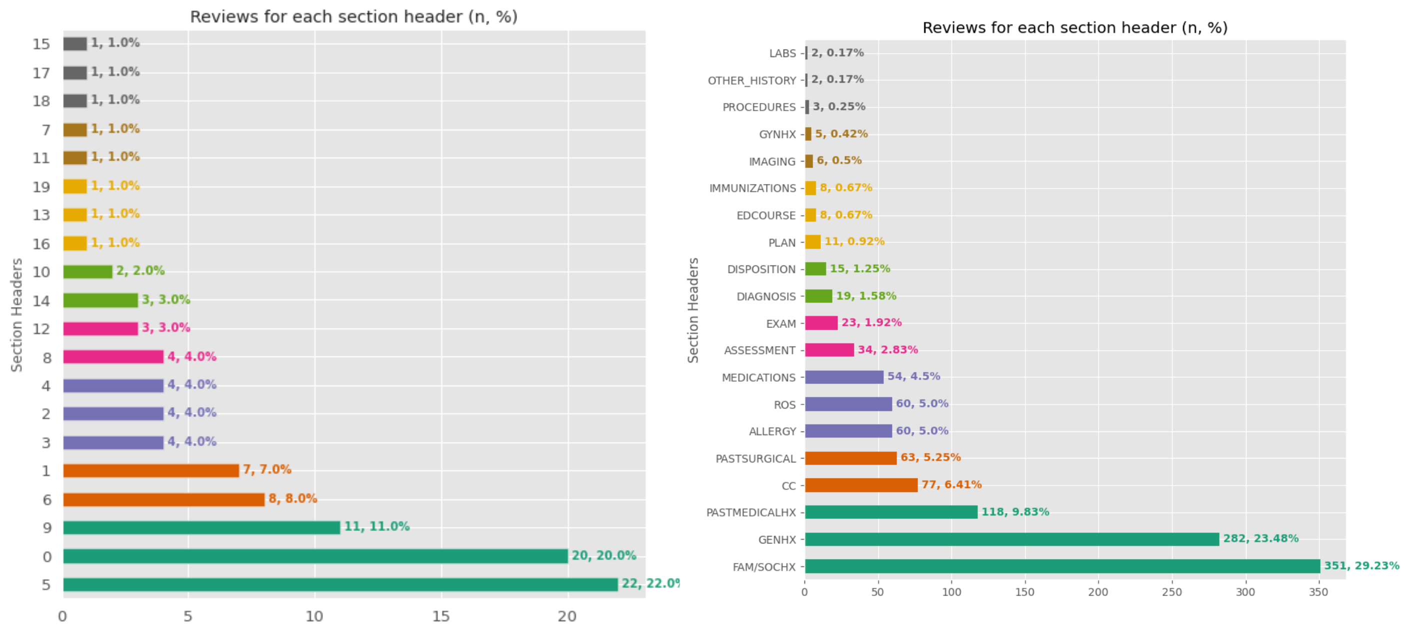
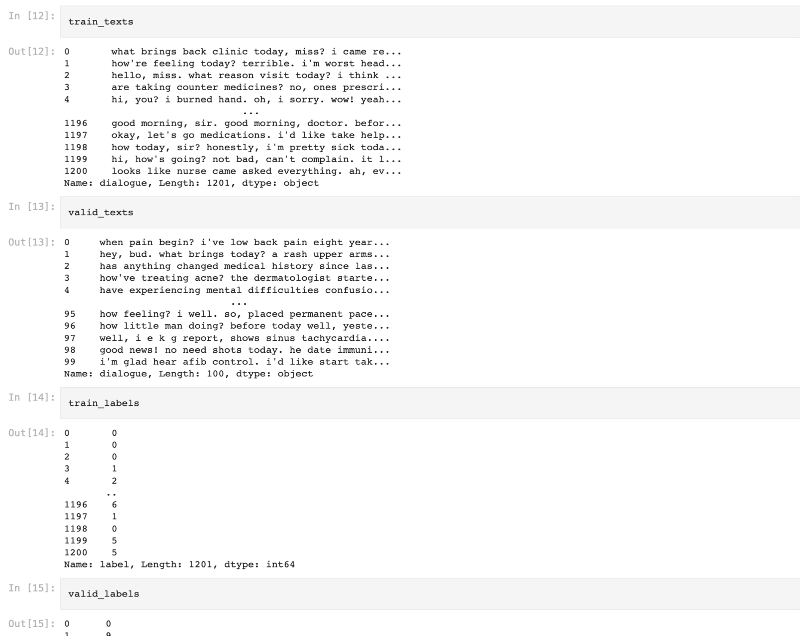
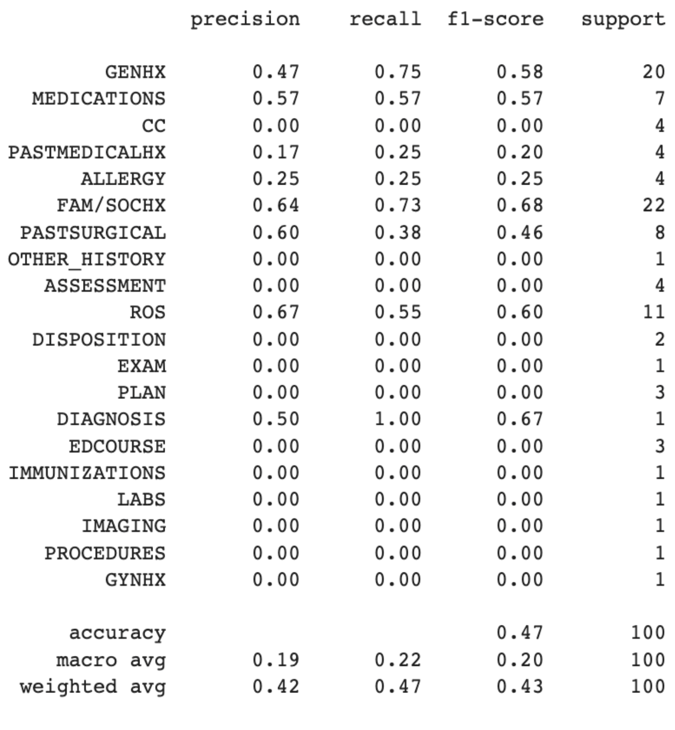


Figure-2: imbalance labels in dataset



```
In [12]: train_texts

Out[12]: 0       what brings back clinic today, miss? i came re...
         1       how're feeling today? terrible. i'm worst head...
         2       hello, miss. what reason visit today? i think ...
         3       are taking counter medicines? no, ones prescri...
         4       hi, you? i burned hand. oh, i sorry. wow! yeah...
                                        ...
         1196    good morning, sir. good morning, doctor. befor...
         1197    okay, let's go medications. i'd like take help...
         1198    how today, sir? honestly, i'm pretty sick toda...
         1199    hi, how's going? not bad, can't complain. it l...
         1200    looks like nurse came asked everything. ah, ev...
         Name: dialogue, Length: 1201, dtype: object

In [13]: valid_texts

Out[13]: 0     when pain begin? i've low back pain eight year...
         1     hey, bud. what brings today? a rash upper arms...
         2     has anything changed medical history since las...
         3     how've treating acne? the dermatologist starte...
         4     have experiencing mental difficulties confusio...
                                      ...
         95    how feeling? i well. so, placed permanent pace...
         96    how little man doing? before today well, yeste...
         97    well, i e k g report, shows sinus tachycardia....
         98    good news! no need shots today. he date immuni...
         99    i'm glad hear afib control. i'd like start tak...
         Name: dialogue, Length: 100, dtype: object

In [14]: train_labels

Out[14]: 0       0
         1       0
         2       0
         3       1
         4       2
                ..
         1196    6
         1197    1
         1198    0
         1199    5
         1200    5
         Name: label, Length: 1201, dtype: int64

In [15]: valid_labels

Out[15]: 0      0
```

Figure-3: data preparation for CRF model



```
                 precision    recall  f1-score   support

         GENHX       0.47      0.75      0.58        20
   MEDICATIONS       0.57      0.57      0.57         7
            CC       0.00      0.00      0.00         4
 PASTMEDICALHX       0.17      0.25      0.20         4
       ALLERGY       0.25      0.25      0.25         4
     FAM/SOCHX       0.64      0.73      0.68        22
  PASTSURGICAL       0.60      0.38      0.46         8
 OTHER_HISTORY       0.00      0.00      0.00         1
    ASSESSMENT       0.00      0.00      0.00         4
           ROS       0.67      0.55      0.60        11
   DISPOSITION       0.00      0.00      0.00         2
          EXAM       0.00      0.00      0.00         1
          PLAN       0.00      0.00      0.00         3
     DIAGNOSIS       0.50      1.00      0.67         1
      EDCOURSE       0.00      0.00      0.00         3
 IMMUNIZATIONS       0.00      0.00      0.00         1
          LABS       0.00      0.00      0.00         1
       IMAGING       0.00      0.00      0.00         1
    PROCEDURES       0.00      0.00      0.00         1
         GYNHX       0.00      0.00      0.00         1

      accuracy                           0.47       100
     macro avg       0.19      0.22      0.20       100
  weighted avg       0.42      0.47      0.43       100
```

Figure-4: evaluation metrics for CRF model

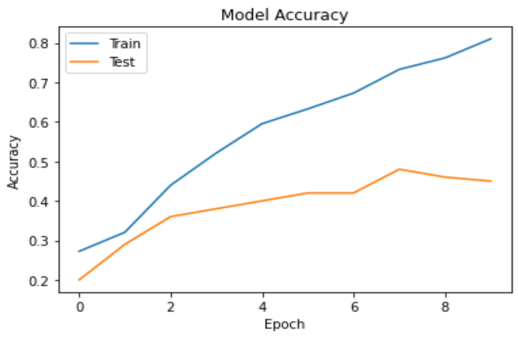


Figure-5: Model accuracy for LSTM (represents 8 epochs out of 21 epochs)

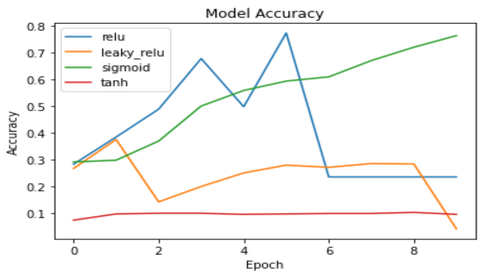
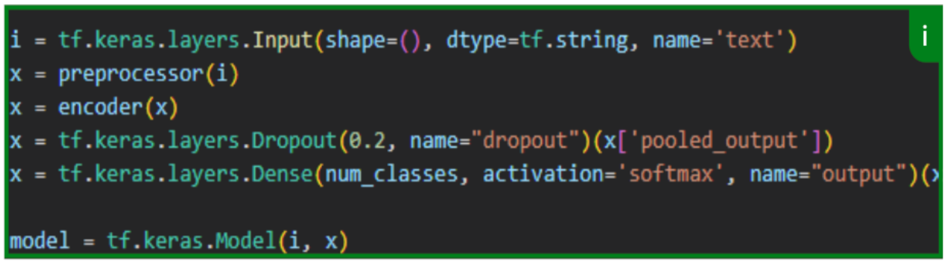


Figure-6: Hyperparameter model for LSTM model


```
i = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
x = preprocessor(i)
x = encoder(x)
x = tf.keras.layers.Dropout(0.2, name="dropout")(x['pooled_output'])
x = tf.keras.layers.Dense(num_classes, activation='softmax', name="output")(x

model = tf.keras.Model(i, x)
```
Figure-7: One hot encoding for Keras layer

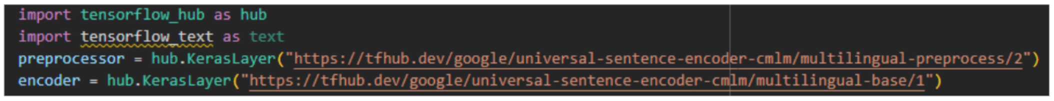
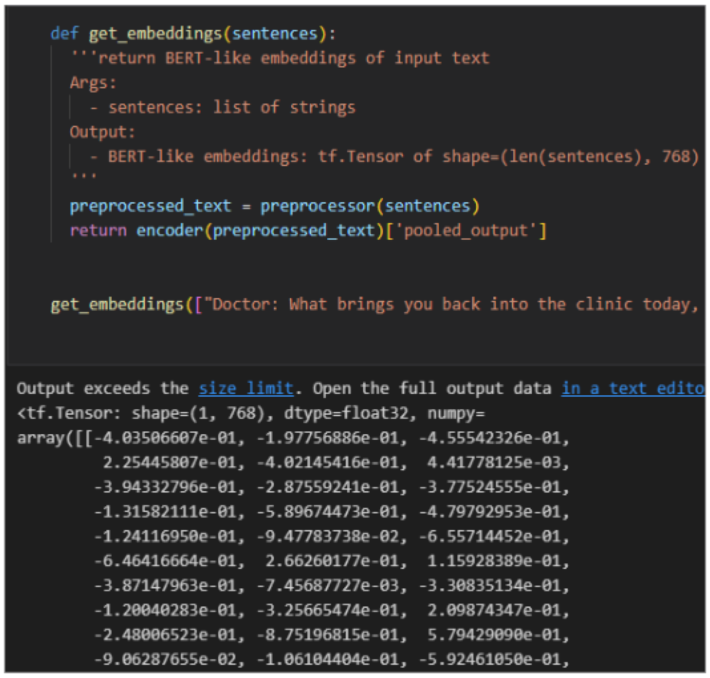

```
import tensorflow_hub as hub
import tensorflow_text as text
preprocessor = hub.KerasLayer("https://tfhub.dev/google/universal-sentence-encoder-cmlm/multilingual-preprocess/2")
encoder = hub.KerasLayer("https://tfhub.dev/google/universal-sentence-encoder-cmlm/multilingual-base/1")
```

Figure-8: mapping input text into a dense vector representation.



```
def get_embeddings(sentences):
  '''return BERT-like embeddings of input text
  Args:
    - sentences: list of strings
  Output:
    - BERT-like embeddings: tf.Tensor of shape=(len(sentences), 768)
  '''
  preprocessed_text = preprocessor(sentences)
  return encoder(preprocessed_text)['pooled_output']


get_embeddings(["Doctor: What brings you back into the clinic today,
```

```
Output exceeds the size limit. Open the full output data in a text edito
<tf.Tensor: shape=(1, 768), dtype=float32, numpy=
array([[-4.03506607e-01, -1.97756886e-01, -4.55542326e-01,
         2.25445807e-01, -4.02145416e-01,  4.41778125e-03,
        -3.94332796e-01, -2.87559241e-01, -3.77524555e-01,
        -1.31582111e-01, -5.89674473e-01, -4.79792953e-01,
        -1.24116950e-01, -9.47783738e-02, -6.55714452e-01,
        -6.46416664e-01,  2.66260177e-01,  1.15928389e-01,
        -3.87147963e-01, -7.45687727e-03, -3.30835134e-01,
        -1.20040283e-01, -3.25665474e-01,  2.09874347e-01,
        -2.48006523e-01, -8.75196815e-01,  5.79429090e-01,
        -9.06287655e-02, -1.06104404e-01, -5.92461050e-01,
```

Figure-9: preprocessing text with pre-processor module

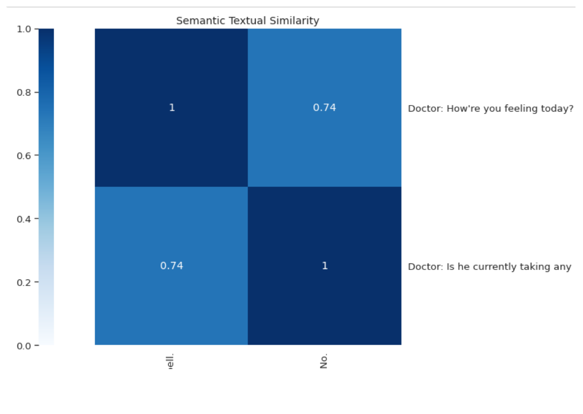
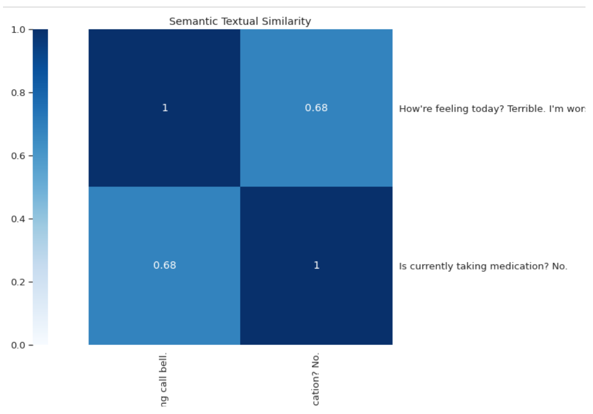
	

Figure-10 Explored semantic similarity between two conversations with cleaning of the data, while feedforward Neural network	*Figure-11* Explored semantic similarity between two conversations without cleaning of the data, while feedforward Neural network

```
              precision    recall  f1-score   support

       GENHX       0.62      0.90      0.73        20
 MEDICATIONS       0.75      0.86      0.80         7
          CC       0.43      0.75      0.55         4
PASTMEDICALHX      0.67      0.50      0.57         4
     ALLERGY       0.67      0.50      0.57         4
   FAM/SOCHX       0.79      1.00      0.88        22
PASTSURGICAL       1.00      1.00      1.00         8
OTHER_HISTORY      0.00      0.00      0.00         1
  ASSESSMENT       0.00      0.00      0.00         4
         ROS       0.75      0.55      0.63        11
 DISPOSITION       0.00      0.00      0.00         2
        EXAM       0.50      1.00      0.67         1
        PLAN       1.00      0.33      0.50         3
   DIAGNOSIS       0.00      0.00      0.00         1
    EDCOURSE       0.00      0.00      0.00         3
IMMUNIZATIONS      1.00      1.00      1.00         1
        LABS       0.00      0.00      0.00         1
     IMAGING       0.00      0.00      0.00         1
  PROCEDURES       0.00      0.00      0.00         1
       GYNHX       0.00      0.00      0.00         1

    accuracy                           0.70       100
   macro avg       0.41      0.42      0.40       100
weighted avg       0.63      0.70      0.65       100
```

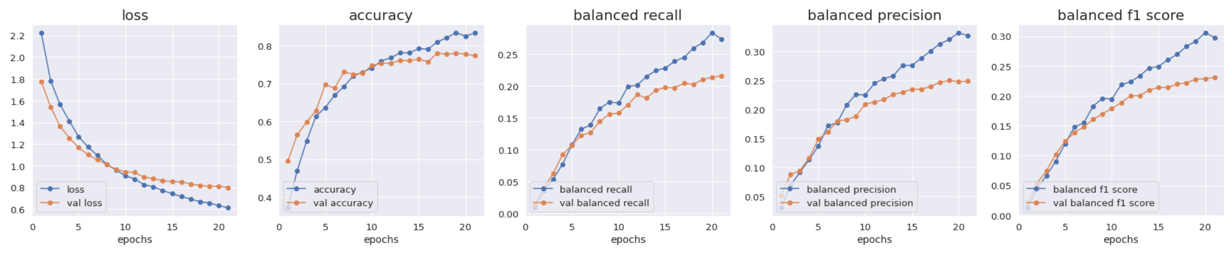


Figure-12: Evaluation Metrics for Feedforward Neural Network

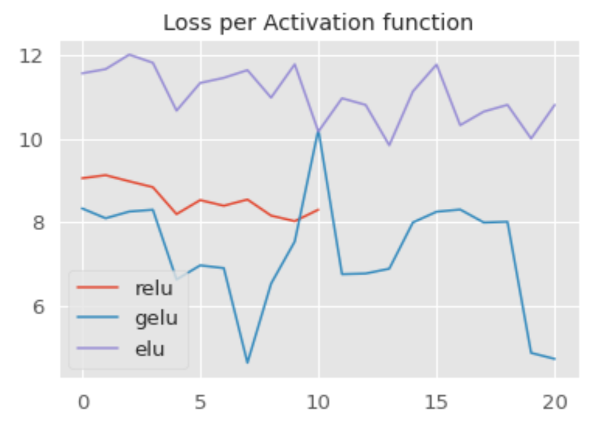
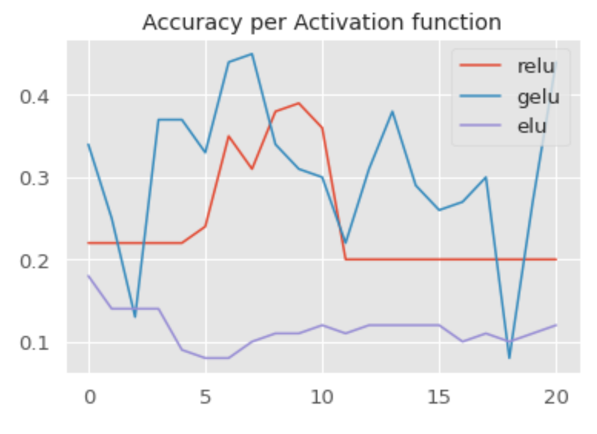
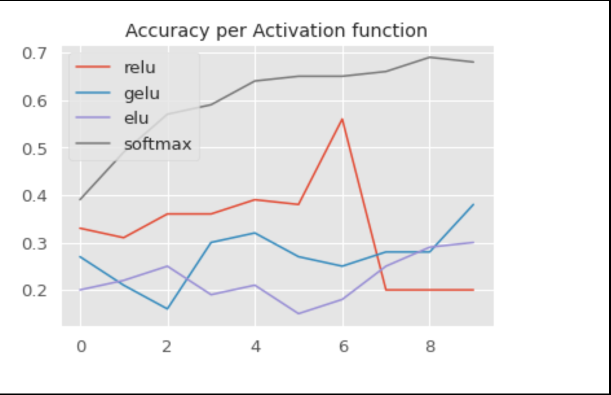
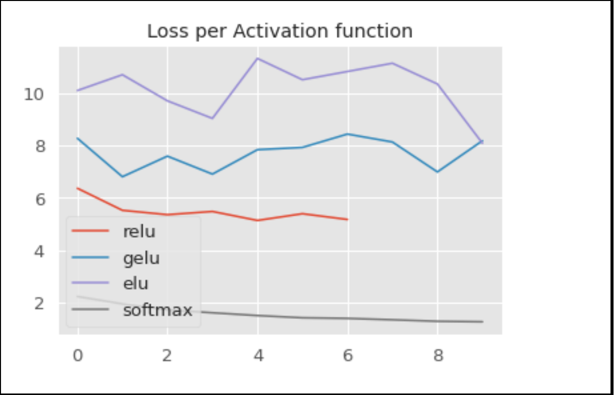
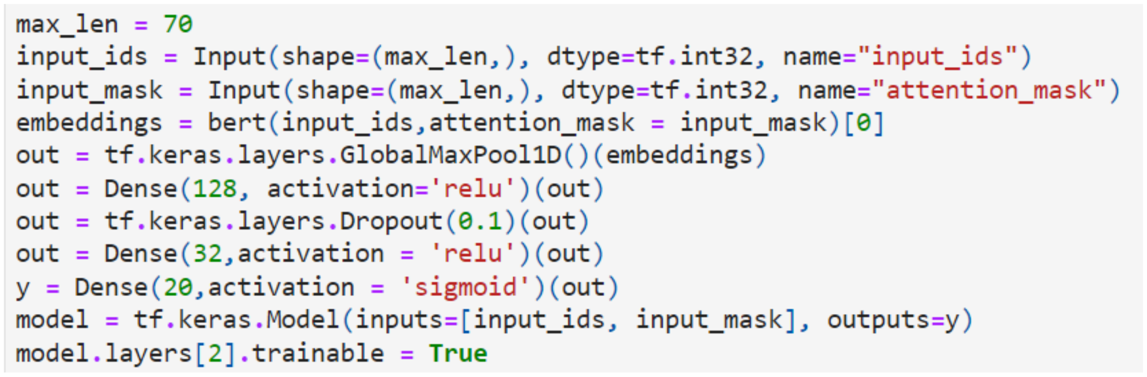
	

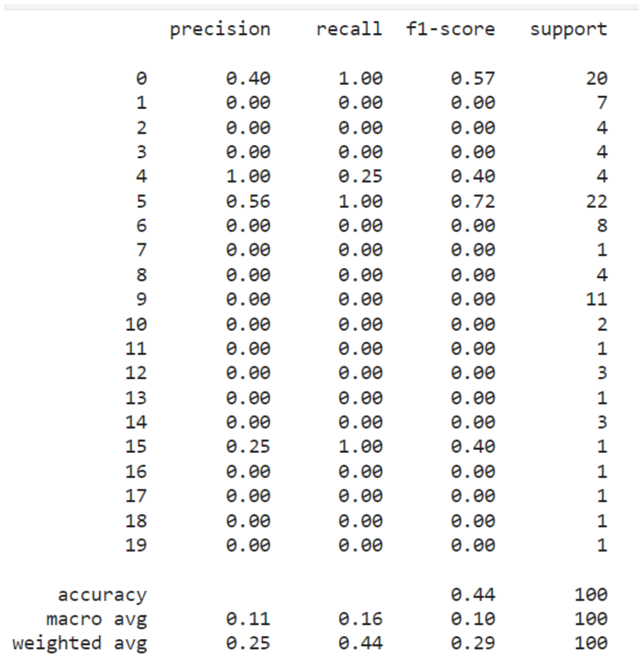
Figure-13 Loss per activation, before 'softmax' in feedforward Neural network model	*Figure-14* Accuracy before 'softmax' in feedforward Neural network model

<table>
<tr><td> </td><td> </td></tr>
<tr><td><i>Figure-15</i>
Accuracy after ‘softmax’ in feedforward Neural network model</td><td><i>Figure-16</i>
Loss per activation, after ‘softmax’ in feedforward Neural network model</td></tr>
</table>



```
max_len = 70
input_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_ids")
input_mask = Input(shape=(max_len,), dtype=tf.int32, name="attention_mask")
embeddings = bert(input_ids,attention_mask = input_mask)[0]
out = tf.keras.layers.GlobalMaxPool1D()(embeddings)
out = Dense(128, activation='relu')(out)
out = tf.keras.layers.Dropout(0.1)(out)
out = Dense(32,activation = 'relu')(out)
y = Dense(20,activation = 'sigmoid')(out)
model = tf.keras.Model(inputs=[input_ids, input_mask], outputs=y)
model.layers[2].trainable = True
```

Figure-17: Defining layers of BERT (cased) model



```
              precision    recall  f1-score   support

           0       0.40      1.00      0.57        20
           1       0.00      0.00      0.00         7
           2       0.00      0.00      0.00         4
           3       0.00      0.00      0.00         4
           4       1.00      0.25      0.40         4
           5       0.56      1.00      0.72        22
           6       0.00      0.00      0.00         8
           7       0.00      0.00      0.00         1
           8       0.00      0.00      0.00         4
           9       0.00      0.00      0.00        11
          10       0.00      0.00      0.00         2
          11       0.00      0.00      0.00         1
          12       0.00      0.00      0.00         3
          13       0.00      0.00      0.00         1
          14       0.00      0.00      0.00         3
          15       0.25      1.00      0.40         1
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         1
          18       0.00      0.00      0.00         1
          19       0.00      0.00      0.00         1

    accuracy                           0.44       100
   macro avg       0.11      0.16      0.10       100
weighted avg       0.25      0.44      0.29       100
```

Figure-18: Metrics for BERT (cased) model

References

Sun, J., Yan, W., Peng, J., Yang, L., & Wei, J. (2020). Automatic topic identification for doctor-patient conversation using machine learning algorithms. Journal of Healthcare Engineering, 2020. https://doi.org/10.1155/2020/8887281

Kim, S., Yoo, S., Kang, S., & Park, S. (2019). Deep learning-based automatic topic identification for doctor-patient conversations. Journal of Healthcare Engineering, 2019. https://doi.org/10.1155/2019/7314071

Anand, A., Li, Y., & Guo, X. (2019). An intelligent topic identification system for doctor-patient conversations. IEEE Access, 7, 131792-131801. https://doi.org/10.1109/ACCESS.2019.2946743

Kim, Y. (2014). Convolutional neural networks for sentence classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1746-1751.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in Neural Information Processing Systems (NIPS), 649-657.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. Proceedings of the 25th International Conference on Machine Learning (ICML), 160-167.

Chetan, K. N., Vijayakumar, P., & Balasubramanian, V. (2021). Identifying the topics in doctor-patient conversations using pre-trained Feedforward Neural Network model. Journal of Healthcare Engineering, 2021. https://doi.org/10.1155/2021/8810303

Zheng, Y., Wang, Y., & Gao, Y. (2020). Automatic classification of topics in Chinese medical consultations with Feedforward Neural Network model. Journal of Medical Systems, 44(10), 194. https://doi.org/10.1007/s10916-020-01625-2

Li, X., Zhang, H., Li, X., Li, Y., Li, R., & Liu, J. (2021). Identifying the topics in Chinese doctor-patient conversations based on Feedforward Neural Network model. Journal of Medical Systems, 45(2), 1-9. https://doi.org/10.1007/s10916-020-01711-5

Lin, J. C. W., Shao, Y., Djenouri, Y., & Yun, U. (2021). ASRNN: A recurrent neural network with an attention model for sequence labeling. Knowledge-Based Systems, 212, 106548.

Tiwari, G., Sharma, A., Sahotra, A., & Kapoor, R. (2020, July). English-Hindi neural machine translation-LSTM seq2seq and ConvS2S. In 2020 International Conference on Communication and Signal Processing (ICCSP) (pp. 871-875). IEEE.

Mishra, N., & Bhatia, M. (2019). A study on neural networks and their application to text classification. International Journal of Engineering and Advanced Technology, 8(6), 2482-2486.

Swati, S., Grobelnik, A. M., Mladenić, D., & Grobelnik, M. (2022). A Commonsense-Infused Language-Agnostic Learning Framework for Enhancing Prediction of Political Polarity in Multilingual News Headlines. arXiv preprint arXiv:2212.00298.

Han, S., Zhang, R. F., Shi, L., Richie, R., Liu, H., Tseng, A., ... & Tsui, F. R. (2022). Classifying social determinants of health from unstructured electronic health records using deep learning-based natural language processing. Journal of Biomedical Informatics, 127, 103984.

Other Resources

https://jalammar.github.io/illustrated-bert/
https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/46_BERT_intro/bert_intro.ipynb
https://github.com/Kent0n-Li/ChatDoctor/blob/main/train.py
https://github.com/topics/multilabel-classification?o=desc&s=updated
https://github.com/DunnBC22/NLP_Projects
https://github.com/DunnBC22/NLP_Projects/blob/main/Multiclass%20Classification/Transformer%20Comparison/Hate%20%26%20Offensive%20Speech%20-%20BERT-Large.ipynb
https://www.youtube.com/watch?v=7kLi8u2dJz0&t=393s&ab_channel=codebasics
https://github.com/christianversloot/machine-learning-articles/blob/main/build-an-lstm-model-with-tensorflow-and-keras.md
https://stackoverflow.com/questions/64689483/how-to-do-multiclass-classification-with-keras
https://huggingface.co/docs/transformers/index
https://towardsdatascience.com/multi-label-text-classification-using-bert-and-tensorflow-d2e88d8f488d