Lesson: Artificial Intelligence II

Fall Semester 2021

Student's name: Christina Christodoulou

I.D Number: LT1200027

Field of Studies: Language Technology

Homework 1

REPORT

In this exercise, a vaccine sentiment classifier is developed using softmax regression analysis in Colab notebook, Python 3.8. Two datasets were provided in csv from containing tweets and the labels 0 for neutral, 1 for anti-vax and 2 for pro-vax. In the following paragraphs, I will explain in detail the steps followed in order to develop a softmax regression model in addition to my observations from running the script.

A. Explanation of script - Steps

First of all, the necessary libraries like 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.

The colab notebook contains ten functions 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, emoticons, numbers and applying lowercasing.

The third function is called *get_columns_vectorize_split(dataframe, feature, label,test_size),* which takes as parameters a dataframe, the name of the feature's column, the name of the label's column and the chosen size of test split. 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. They are turned into vectors using the *TF-IDF Vectorizer*. *TF-IDF Vectorizer* is set to apply lowercase, tokenize using the nltk package, remove

English stopwords and create unigrams. I also experimented with *CountVectorizer* applying the same parameters as in *TF-IDF Vectorizer*. The input and target variables are then split into training and testing set using the *train_test_split* function offered by sklearn.

The fourth function is called *plot_target_values(csv_file,target_label)*, which takes as parameters a file in csv form, the name of the feature's column and the name of the label's column. It produces a plot of the counts of the target variable. This was the function that I observed the data imbalance problem.

The fifth function is called *search_best_parameters(X_train,Y_train)*, which takes as parameters the split X_train and Y_train sets and initializes a logistic regression model with class weight *balanced*, 1000 max iterations and uses the GridSearchCV method, various C parameters as input in order to find the best C parameter to be used for the following model. I created this function in hope to assist me in tuning my basic softmax regression model. I aimed at producing good scores, while trying to solve the class imbalance problem that was observed. The downside of this function was that the cell took longer time to run than the rest.

The sixth function is called *train(X_train,Y_train)*, which takes as parameters the split X_train and Y_train values, uses the best C parameter from the previous function to train a softmax regression classifier as well as plots learning curves to check overfitting or underfitting issues and saves the trained model in pickle. The softmax regression classifier is initialized with a 0.1 C parameter, class weight *balanced* and 1000 maximum number of iterations.

The seventh function is called *predict_evaluate_classifier(X_train, X_test, Y_train, Y_test,saved_model),* which takes as parameters all the split X and Y values as well as the trained model in pickle. It fits the new unseen data to the trained model and makes predictions. Moreover, it includes evaluation of the model by calculating the accuracy, precision, recall and f-measure as well as prints the confusion matrix. Finally, it plots learning curves to check overfitting or underfitting issues.

The eight function is called *find_best_parameters_model(csv_file,feature,label)* and includes three of the functions described above, the *read_explore_dataframe(csv_file)*,the *get_columns_vectorize_split(df,feature,label,test_size)* and the *search_best_parameters(X_train,Y_train)*. Each function takes as parameters the output of the previous. It is only used as a test to find the optimal C parameter and returns a logistic regression model.

The ninth function is called *data_train_model(csv_file,feature,label)* and is similar to the eight function, since it includes the first two same functions as well as the *train(X_train,Y_train)* function. It returns the mean and standard deviation of the train and test scores that were computed to plot the learning curves. The eighth and ninth functions are only used on the training dataset.

The tenth function is called *data_evaluate_model(csv_file,feature,label,saved_model)*, which also includes the first two functions as in the eight and ninth function, but the third function is the one that loads the trained model in pickle, makes predictions using the validation dataset and evaluates the model. This function returns the classification matrix produced in the last inside function. It is only used on the validation dataset and

is meant to be used on the test set by simply changing the name of the *csv_file* parameter.

B. Steps

I ran all the cells to initialize each function. During the training phase, the functions that run using the training dataset were the *find_best_parameters_model()* and the data_train_model() along with the *tweet* and *label* as features. The function plot_target_values() was also run using the training dataset and the *label*. Next comes the validation phase, where the functions plot_target_values() and data_evaluate_model() run, using the validation dataset, the *tweet* and the *label*.

C. Observations

Checking the distribution of the three classes (neutral:0, anti-vax:1, pro-vax:2) in the training and in the validation datasets, it is evident that label 1 is a minority class, which constitutes a data imbalance problem. In the *vaccine_train_set.csv*, class 0 contains 0.466825, class 2 contains 0.403418, while class 1 only 0.129757 values. In the *vaccine_validation_set.csv*, class 0 contains 0.466696, class 2 contains 0.403593, whereas class 1 only 0.129711 values. For this reason, I added the class_weight parameter as *balanced* in the two logistic regression models and resampled to achieve data balance and fair treatment of the model for each class during making predictions on the validation and test sets.

An example sentence before preprocessing:

"Sip N Shop Come thru right now #Marjais #PopularNobodies #MMR 🤲🤲🤲🤲 @ Marjais SipNShop http://t.co/JfWAH7uzWE"

The sentence after pre-processing:

"sip n shop come thru right now marjais popularnobodies mmr marjais sipnshop"

I split the data into 80% training and 20% testing set and run 2 models with different vectorizers. From the board below, it is evident that the model using CountVectorizer achieved the best results in all the metrics compared to the model using TF-IDF Vectorizer despite the same number of features. Both models make good predictions with the data imbalance problem solved.

Vectorizer	Number of features - Training	Accuracy	Precision (class 0,1,2)	Recall (class 0,1,2)	F-measure (class 0,1,2)
TF-IDF	21400	83.21%	82.5%, 83.6%, 83.3%	78.4%, 89.3%, 81.4%	80.4%, 86.3%, 82.3%
CountVectorizer	21400	92.48%	93.3%, 93.6%, 90.5%	88.5%, 97.8%, 90.7%	90.8%, 95.7%, 90.6%

The number of True Positive predictions (correct label prediction for the correct class) is very satisfactory for both models:

TF-IDF

Class 1: 335 out of 427

Class 2: 419 out of 469

Class 3: 386 out of 474

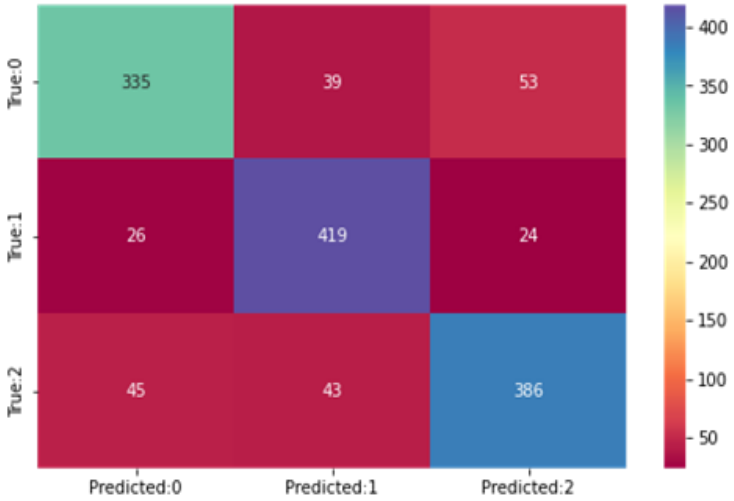
CountVectorizer

Class 1: 378 out of 427

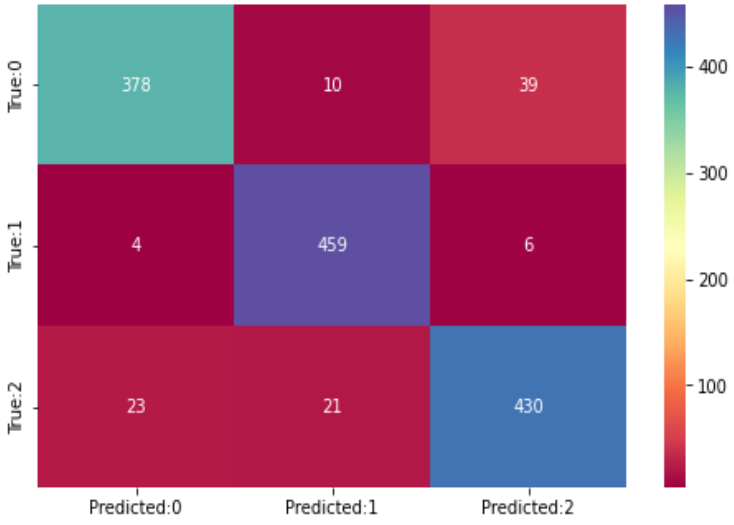
Class 2: 459 out of 469

Class 3: 430 out of 474

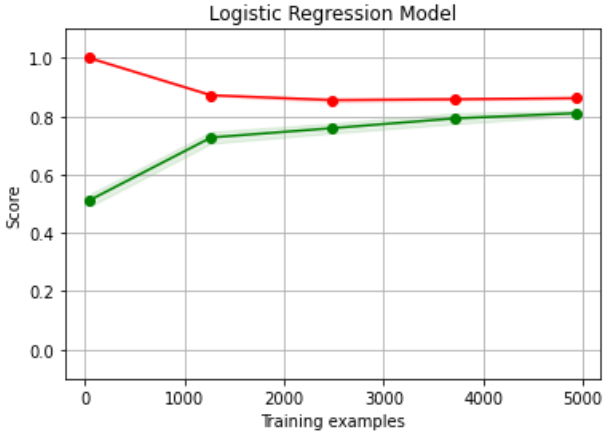
TF-IDF Vectorizer Classification Matrix



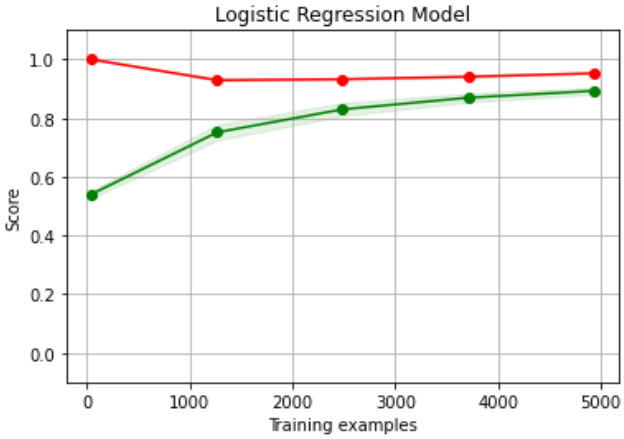
CountVectorizer Classification Matrix



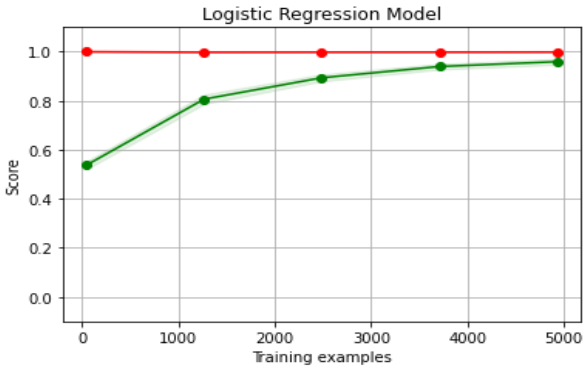
TF-IDF Vectorizer learning curve (validation dataset)



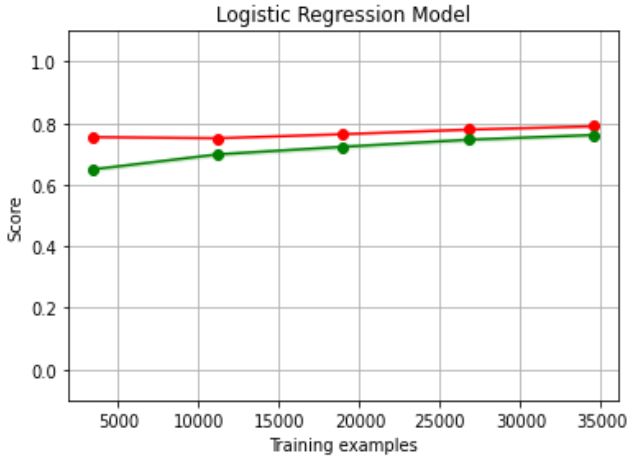
CountVectorizer learning curve (validation dataset)



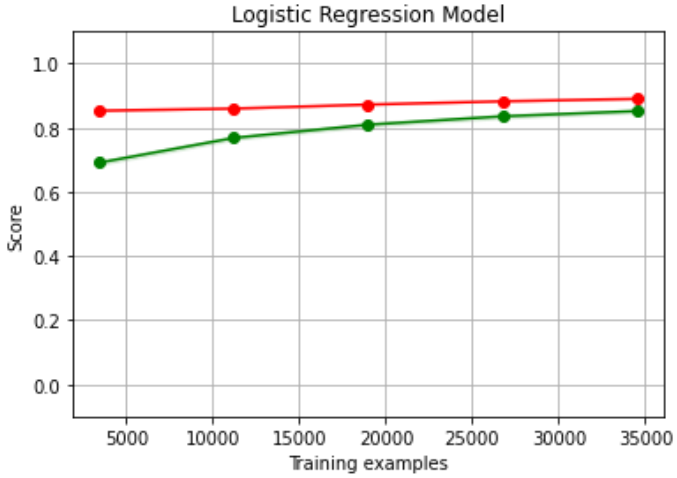
For training example size less than 2500, there is a bigger gap between the training score (red line) and the validation score (green line), which means that the model has not yet learned the data. After 2500 training examples, the gap starts to gradually become smaller between the lines. The lines gradually increase and are close to converge. It seems that the model is learning well the training data and is generalizing well without underfitting or overfitting issues. There is not high model variance. The learning curves in the CountVectorizer model's plot reach higher score, which denotes better model performance. At first, I used the C parameter from the GridSearch (C:10), but I noticed that the model overfitted, so I selected to change the C parameter from 10 to 0.1.



TF-IDF Vectorizer learning curve (train dataset)



CountVectorizer learning curve (train dataset)



From the two plots of learning curves, it is shown that after 2000 training examples the training and the validation line increase gradually and come very close, especially in the first plot, where they converge at the end. This means that both models were able to learn from the data, have a good bias-variance trade-off and achieved a high score without underfitting or overfitting.