Project Progress Report of Emotion Classification using Machine Learning on a Multi-Class Dataset

Members:

- - Carlos Rabat: czr0067@auburn.edu, *Project Coordinator*
- - Samarth Kumar: szk0187@auburn.edu
- - Jeronime Houndonougbo: jah0268@auburn.edu

1. Data Set and Preprocessing

1.1 Data Set: Training and Testing Data

The dataset contains a total of 416,809 samples, so they were partitioned into 333,447 training samples and 83,362 testing samples, meaning that 80% of the data was allocated towards training the model while 20% of the data was allocated for testing. Since the original dataset only consists of two columns (text and label), we used TD-IDF and Bag of Words (BoW) approaches to extract 1000 features from the raw data. These features represent the 1000 most common words.

1.2 Metrics

To evaluate our models, we used accuracy to measure the overall performance of each model. Our main focus for now is accuracy, but once we reach a high level of accuracy for all models, we will then utilize other metrics such as precision, recall, and f1-score for a more thorough analysis. For these, we may also utilize confusion matricies, precision-recall curves, and classification reports.

1.2 Data Preprocessing

Because our dataset primarily involves text, we must process the data and extract numerical value for each string to be able to fit the data into the machine learning models. First, we needed to preprocess the dataset. We began by tokenizing the texts, removed the stop words, set each letter to lowercase, and stemmed the tokens. Once we tokenized the data, we partitioned the data into testing and training samples. Next, we needed to convert the textual data into numerical values, so we used the Bag of Words (BoW) technique. CountVectorizer is a class within the Scikit-learn library that allows for converting documents of text into numerical matrices. We stored lists of the strings from our X_train and X_test samples. We fit these values into our vectorizer and set max_features to 1000 to limit the attributes to the top 1000 words. After using the vectorizer for the BoW approach, the data preprocessing was complete and ready to use in the machine learning models.

2. Machine Learning Algorithms

2.1 Random Baseline

To measure the performance of our machine learning we created a simple model that predicts randomly uniformly between the 6 classes, the average performance of the model was with an accuracy of 14.58%. This means that any of our models predicting less than this will be considered completely ineffective.

2.2 Multi-Layer-Perceptron

For the multi-layer perceptron various structures with different number of layers, number of neurons in each layer and different types of regularization were attempted to maximize the performance of this models. Three

different models are displayed increasing in complexity. All models were trained on the same training and test data with the Adam optimizer. They also shared other hyperparameters such as validation split of 20%, batch size of 64 and 15 epochs. The current model accuracies are the following:

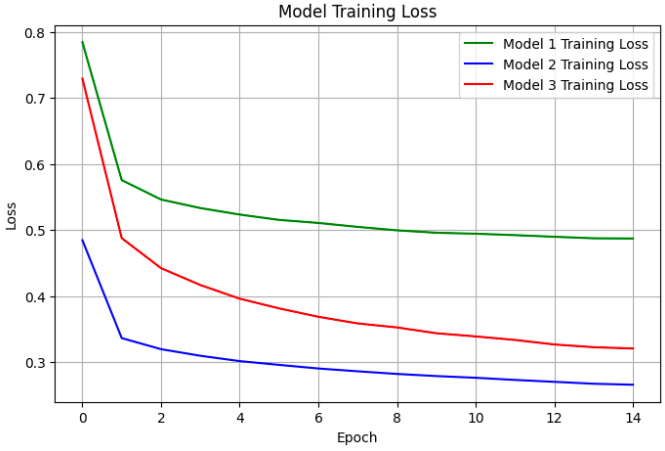
- Model 1: 41%
- Model 2: 39%
- Model 3: 40%

The model structure that we see below (Model 1) has the best accuracy that was achieved using the MLP algorithm so far:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	16,016
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 8)	136
dropout_1 (Dropout)	(None, 8)	0
dense_2 (Dense)	(None, 6)	54

2.2.1 Training Loss

Below is the training loss against each epoch of the three models implemented. After epoch ten, the loss doesn't change as much meaning each model is reaching convergence. In the section about difficulties is more information on why the MLP is our worst performing model at the moment.

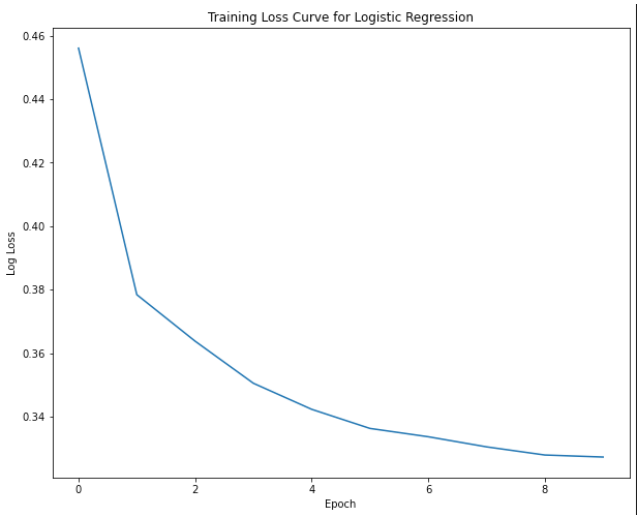


2.3 Logistic Regression

For the Logistic Regression model, we used the model imported from the Scikit-learn library. Additionally, we utilized the Stochastic Gradient Descent Classifer (SGDClassifer) library from Scikit-learn for the training loss curve. For the model, we set max_iterations to 500 to ensure convergence and we used the default values for the other hyperparameters. The Logistic Regression Model predicted the classes with an accuracy of approximately 86.56%.

2.3.1 Training Loss

Below is the Training Loss Curve for the Logistic Regression Model . (Used a maximum of 10 epochs since it is better to use fewer epochs for larger data sets.



2.4 Naive Bayes Classifier

The code loads training and testing data from CSV files, tokenizes the text data, and converts it into bag-of-words representations using CountVectorizer. A Naive Bayes Classifier is implemented, which trains on the training data by updating class and word counts. The classifier then predicts the emotions of test instances based on the trained model and calculates probabilities using the Naive Bayes formula. The accuracy of the classifier is evaluated by comparing the predicted and true labels of the test instances. Finally, a confusion matrix is computed to provide a detailed breakdown of the predictions and actual classes. The overall accuracy of the Naive Bayes Classifier is 0.8674, which means it correctly classified 86.74% of the test samples.

3. Challenges Faced

When working with the Logistic Regression model, we had difficulties in evaluating the training loss curve. While metrics such as accuracy, precision, etc. , are easily imported within the scikit-learn library and work with any imported model, determining the training loss curve required additional steps. After reading the documentation for scikit-learn, we found that we could implement an SGDClassifier and use log_loss to specifically work with Logistic Regression, to display the training loss curve overall. Furthermore, our next goal is to figure out how to hypertune the parameters so that we can improve the accuracy of the Logistic Regression model.

When working with The Naive Bayesian Classifier, one of the challenges was the Handling of Categorical Data: Naive Bayes classifiers typically work well with categorical features. To handle that, have converted the tokenized texts into strings using ' '.join(tokens) before applying the CountVectorizer. This step allows the classifier to treat the data as categorical, as expected for Naive Bayes algorithms.

For Multi-Layer-Perceptron Algorithm the main challenge is to find the right structure. Unlike the other algorithms, MLP have more variation because we need to pick the right number of layers, number of neurons in a layer and regularization per layer. Moving forward we will try to optimize the MLP structure to have an accuracy similar to the other algorithms such as Naive Bayes and Logistic Regression.