Data Mining [CSE-572]

Homework 2

Name:- Lalith Abhiram Sasi Bhushan Velampati

ASU ID:- 1229176470

ASURITE: - lvelampa

This document furnishes essential descriptions corresponding to all the answers found in the. ipynb file.

Question 1:- Preprocessing the raw Training data.

The initial preprocessing steps involved the following: -

• **Lowercasing**: Ensures uniform text formatting by converting all letters to lowercase, aiding in consistent analysis.

• **Punctuation Removal**: Simplifies text by eliminating punctuation marks, focusing on essential content for analysis.

• **Tokenization**: Divides text into units (tokens) for foundational analysis, enabling tasks like word frequency and syntactic parsing.

• **Stop word Removal**: Filters out common, low-information words, sharpening focus on meaningful content for tasks like sentiment analysis.

• **Stemming**: Reduces words to their base form for standardized representation, enhancing analysis in tasks like information retrieval.

• **N-grams**: Provides context by considering word sequences, crucial for tasks like language modeling and coherent text generation. Converted the Initial raw training data into unigrams and Bigrams as a part of n-grams modelling but only will be using Unigrams for the classification techniques as a part of this Assignment.

You can verify the displayed Train and Test data frames in the notebook for the correct representation of Unigrams and Bigrams.

The function extract_keywords, which uses TF-IDF (Term Frequency-Inverse Document Frequency) to extract a specified number of keywords from a document. It then provides an example of how to use this function and demonstrates the extraction of keywords for multiple documents. The results, containing the top keywords for each document, are stored in all keywords list

Question 1.a) :- Run Neural Network with 2 hidden layers along with Count Vectorizer()

A Multi-Layer Perceptron (MLP) Classifier with 128 units in two hidden layers to classify text data. It utilizes a CountVectorizer to convert text into a numerical format suitable for machine learning. The cell evaluates the model's performance using 5-fold cross-validation, providing insights into its generalization capabilities.

For each fold, the data is split into training and validation sets. The pipeline, integrating CountVectorizer and MLP Classifier, is trained on the former. The script records training and validation accuracies, key indicators of the model's learning ability. It also computes the mean and standard deviation of these accuracies, offering a concise summary of the model's overall performance and consistency.

Question 1.b) :- Feature extraction using TFIDF and Glove

The approach combines TF-IDF features with pre-trained GloVe word embeddings to perform text classification using a Multi-Layer Perceptron (MLP) Classifier. The code first loads GloVe embeddings and initializes a TF-IDF vectorizer to convert text data into numerical format. It then computes the average GloVe embeddings for each sentence.

The TF-IDF and GloVe features are combined and used to train an MLP Classifier. The subsequent manually performs 5-fold cross-validation for better evaluation. Training and validation accuracies are recorded for each fold. Finally, the mean and standard deviation of the validation accuracies are calculated.

This approach leverages the strengths of both TF-IDF and GloVe embeddings, potentially enhancing the model's ability to capture semantic information in text data.

Question 1.c) :- Description of generated Features

Features are generated using the bag-of-words approach with CountVectorizer. This method converts text data into numerical format by creating a matrix where each row represents a document, and each column represents a unique word in the entire corpus. The value in each cell indicates the frequency of a word in a particular document.

The pipeline combines this feature extraction step with an MLP (Multi-Layer Perceptron) Classifier. The MLP is a type of artificial neural network capable of learning complex patterns in data. In this case, it's used to classify text data into different categories.

The code then implements 5-fold cross-validation. This technique divides the data into five subsets or "folds". The model is trained on four of these folds and evaluated on the remaining one. This process is repeated five times, ensuring that each fold serves as both a validation and training set.

In summary, this code snippet efficiently combines feature extraction using bag-of-words with a powerful MLP Classifier, assesses the model's performance using 5-fold cross-validation, and reports accuracy metrics for each fold.

Question 1.d) :- Report the accuracy and Standard deviation

The table compares text classification methods. CountVectorizer and GloVe+TFIDF achieve perfect training accuracy. GloVe+TFIDF yields the highest validation accuracy (97.3%), outperforming CountVectorizer (97.1%). TFIDF stands at 96.5%, while GloVe alone scores 94.4%, indicating its lower efficacy.

Question 1.e) :- Visualize the data in Bar Chart

A bar chart has also been included in the Notebook for better visualizations

Question 2:- Run Neural Networks with Learning Rates

Question 2.a):- Parameter Settings

Learning Rates:

learning_rates = [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1]: A list of candidate learning rates is defined. These learning rates will be tested to find the one that yields the best results.

MLP Classifier:

hidden_layer_sizes=(128, 128): This sets the architecture of the neural network. It has two hidden layers with 128 neurons each.

max_iter=4000: Specifies the maximum number of iterations for training the MLP. This helps prevent training from continuing indefinitely.

Cross-Validation:

kf = KFold(n_splits=5, shuffle=True, random_state=42): K-Fold Cross-Validation is used with 5 folds. This means the data is divided into 5 subsets, and the model is trained and evaluated 5 times.

Nested Loops:

The outer loop iterates over the candidate learning rates.

The inner loop performs the cross-validation process.

Data Splitting:

X_combined[train_index], X_combined[val_index], y.iloc[train_index], and y.iloc[val_index]: These lines split the data into training and validation sets using the indices provided by the KFold object. X_combined likely contains the preprocessed feature data.

Model Training and Evaluation:

mlp_classifier.fit(X_train, y_train): The MLP Classifier is trained on the training data.

mlp_classifier.score(X_train, y_train) and mlp_classifier.score(X_val, y_val): These lines compute the training and validation accuracies, respectively.

Results Storage:

results.append([...]): For each learning rate, the mean and standard deviation of training and validation accuracies are calculated and stored in the results list.

Results DataFrame:

results_df = pd.DataFrame(results, columns=['Learning Rate', 'Avg Training Accuracy', 'Std Training Accuracy', 'Avg Validation Accuracy', 'Std Validation Accuracy']): This line creates a Pandas DataFrame to store the results, making it easy to analyze and visualize the performance of different learning rates.

Question 2.b.1):- Report the Accuracies and Standard Deviations

The table presents the impact of varying learning rates on an MLP Classifier's performance. Notably, all rates achieve perfect training accuracy, suggesting potential overfitting. Among these, a learning rate of 0.0001 yields the highest average validation accuracy (97.5%) with low variability (std = 0.0038). Generally, the model demonstrates robustness to different learning rates, as indicated by consistent high performance. Notably, the rates of 0.001 and 0.0003 also exhibit strong validation accuracies, both around 97.3%. However, the experiment underscores the importance of fine-tuning hyperparameters to ensure optimal model performance.

Question 2.b.2):- Visualize with a line Graph

A line figure has also been included in the notebook for better visualization

Question 2.c):- Neural Networks with Optimizers:-

This cell implements a text classification task using a simple neural network in PyTorch. The network is trained with three different optimizers (SGD, Adam, RMSprop) over 5-fold cross-validation. The data is represented using a bag-of-words approach.

A custom neural network class is defined, consisting of a linear layer. The network is trained with various optimizers, and accuracy scores are recorded for both the training and validation sets. The experiment aims to compare the performance of different optimizers in this classification task.

The results are stored in a list, including the optimizer used, along with the corresponding training and validation accuracies. This information will be further analyzed to determine the most effective optimizer for this specific task.

In summary, the code performs a thorough evaluation of different optimizers on a text classification problem, providing valuable insights into their impact on model performance.

Question 2.c.1):- Report the Accuracies and Standard Deviations

The table displays the performance of different optimizers (SGD, Adam, RMSprop) in a text classification task using a neural network. Notably, RMSprop consistently achieves the highest validation accuracies, reaching up to 98.6%. Adam follows closely with strong validation scores averaging around 96.5%. On the other hand, SGD exhibits lower validation accuracies, averaging approximately 87.8%. Training accuracies are consistently higher than validation, suggesting potential overfitting. Overall, RMSprop demonstrates superior performance in this context, while Adam also proves effective. This experiment highlights the critical role of optimizer choice in optimizing model performance for text classification.

Question 2.c.2):- Visualization in Bar Graph

A line figure has also been included in the notebook for better visualization.

Question 3.a):- The initial preprocessing steps involved the following

• **Lowercasing**: Ensures uniform text formatting by converting all letters to lowercase, aiding in consistent analysis.

• **Punctuation Removal**: Simplifies text by eliminating punctuation marks, focusing on essential content for analysis.

• **Tokenization**: Divides text into units (tokens) for foundational analysis, enabling tasks like word frequency and syntactic parsing.

• **Stop word Removal**: Filters out common, low-information words, sharpening focus on meaningful content for tasks like sentiment analysis.

• **Stemming**: Reduces words to their base form for standardized representation, enhancing analysis in tasks like information retrieval.

• **N-grams**: Provides context by considering word sequences, crucial for tasks like language modeling and coherent text generation.

Question 3.b :- Discussing the parameters chosen for the Model

This cell snippet begins by preprocessing the 'Unigrams' column in the 'test' DataFrame, applying a TF IDF transformation to obtain feature vectors in 'X_test_tfidf'.

Next, a MLP Classifier is initialized and trained on the entire training data, utilizing the following hyperparameters:

optimizer: adam

learning_rate: 0.0001

These parameters have achieved the highest accuracy previously. Therefore, will be using it for predicting the labels in the test dataset. The model is then used to predict labels for the test data, and these predictions are added as a new column 'Predicted Category' in the 'test' DataFrame.

Finally, the updated 'test' DataFrame, now including the predicted categories, is printed.

Question 3.c: -Discussing the Performance of the chosen model for the training Data

Predicts labels for the training data using the trained MLPClassifier. Calculates the accuracy of the predictions compared to the actual training labels.

Prints the training accuracy. The Accuracy achieved was 0.97.

Adds a new column 'Predicted Category' with the predicted labels to the 'train' dataFrame.

Prints the updated 'train' DataFrame with predicted categories for the training set.

****The new updated column can be visualized in the notebook, under "Predicted Category.**

Question 3.d:-

The predicted labels have been added as a new column in the test dataframe and are saved in the labels.csv file which is submitted along with this description document.

The labels.csv has no header and ArticleID and Predicted Labels in different columns as specified