1. FastText Model:

A/ Introduction:

FastText is a tool for text classification. It learns to associate text with labels, allowing it to predict the label for new text:

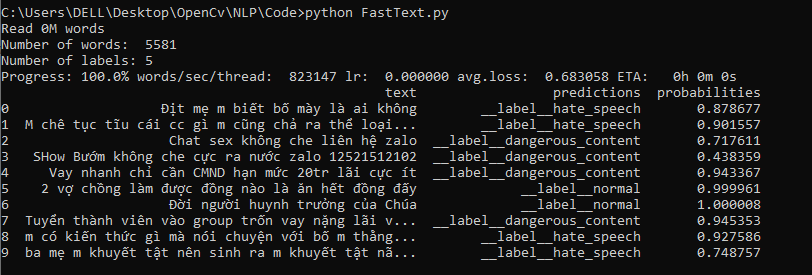
* Word Embeddings: FastText learns about the meaning of words by representing them as vectors in a space where similar words are close together.
* Subword Information: It considers not just whole words but also parts of words, enabling it to understand word morphology better.
* Classification: FastText predicts labels for text by learning patterns in the data during training.

B/ Implementation:

Here is how we implement FastText to our script:

* Data Preprocessing: The script prepares the data for training by cleaning it and splitting it into training and testing sets.
* Model Training: FastText is trained on the prepared data, learning to associate text with labels.
* Testing and Evaluation: The trained FastText model is used to predict labels for the test data, and the predictions are evaluated.

C/ Result:



2. Logistic Regression Model:

A/Introduction:

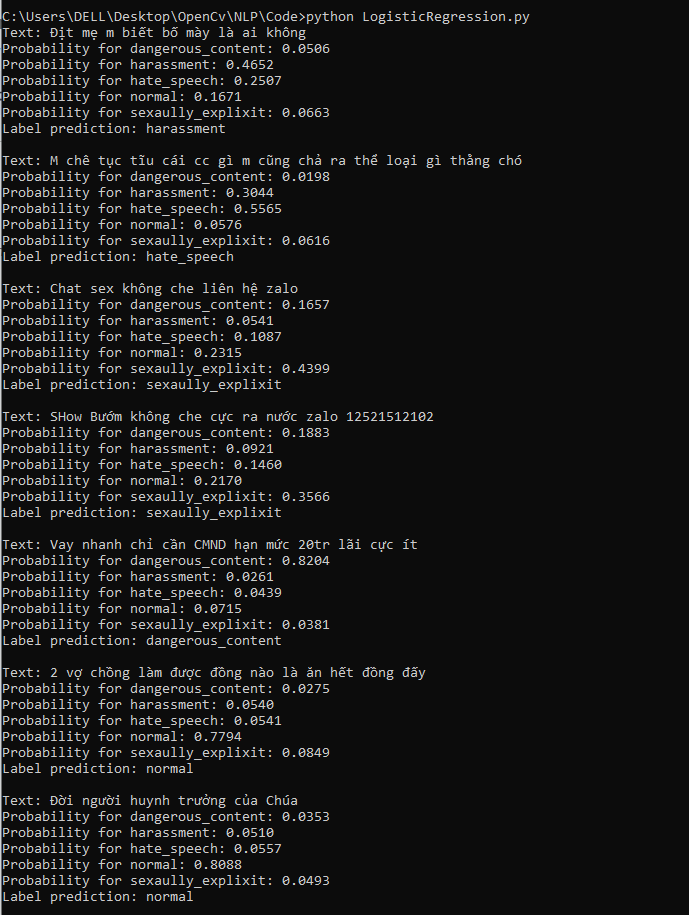
Logistic Regression is a basic yet effective algorithm for classification tasks:

* Linear Model: It learns a linear relationship between the input features (TF-IDF values for words in this case) and the output labels.
* Sigmoid Function: The output of the linear model is passed through a sigmoid function, squashing the output between 0 and 1, representing probabilities.
* Classification: Based on these probabilities, Logistic Regression predicts the most likely class for each input.

B/ Implementation:

Here is how we implement Logistic Regression to our script:

* Data Preprocessing: The script prepares the data for training by cleaning and splitting it.
* Model Training: Logistic Regression is trained on the prepared data, learning to classify text based on TF-IDF values.
* Testing and Evaluation: The trained Logistic Regression model is used to predict labels for the test data, and the predictions are evaluated.

C/ Result:  


3. Multinomial Naive Bayes Model:

A/Introduction:

Naive Bayes is a probabilistic classifier based on Bayes' theorem with a "naive" assumption of feature independence:

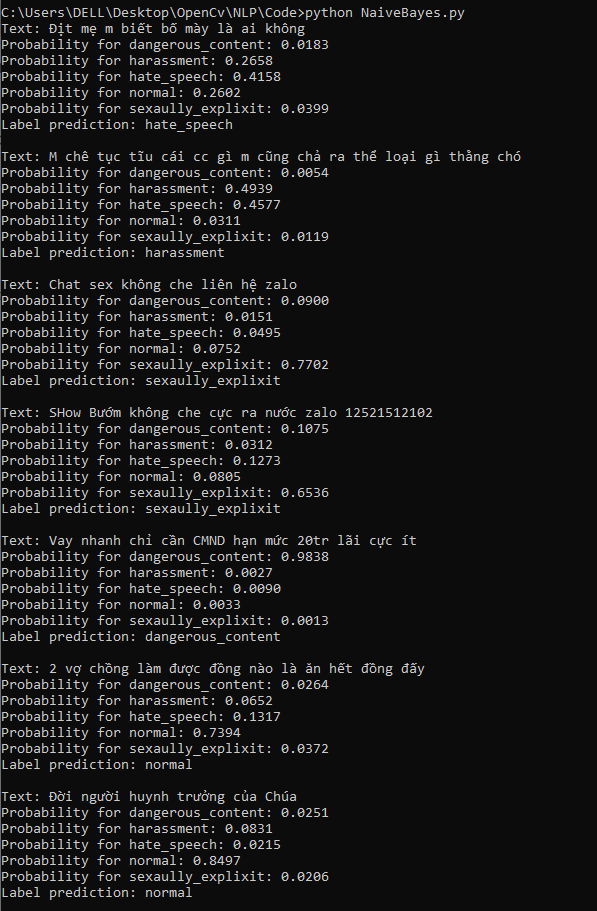
* Probability Model: It models the probability of a class given the input features.
* Feature Independence: It assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
* Classification: Based on these probabilities, Naive Bayes predicts the most likely class for each input.

B/Implementation:

Here is how we implement Naive Bayes to our script:

* Data Preprocessing: The script prepares the data for training by cleaning and splitting it.
* Model Training: Multinomial Naive Bayes is trained on the prepared data, learning to classify text based on TF-IDF values.
* Testing and Evaluation: The trained Naive Bayes model is used to predict labels for the test data, and the predictions are evaluated.

C/Result:



4. Random Forest Classifier:

A/Introduction:

Random Forest is an ensemble learning method based on decision trees:

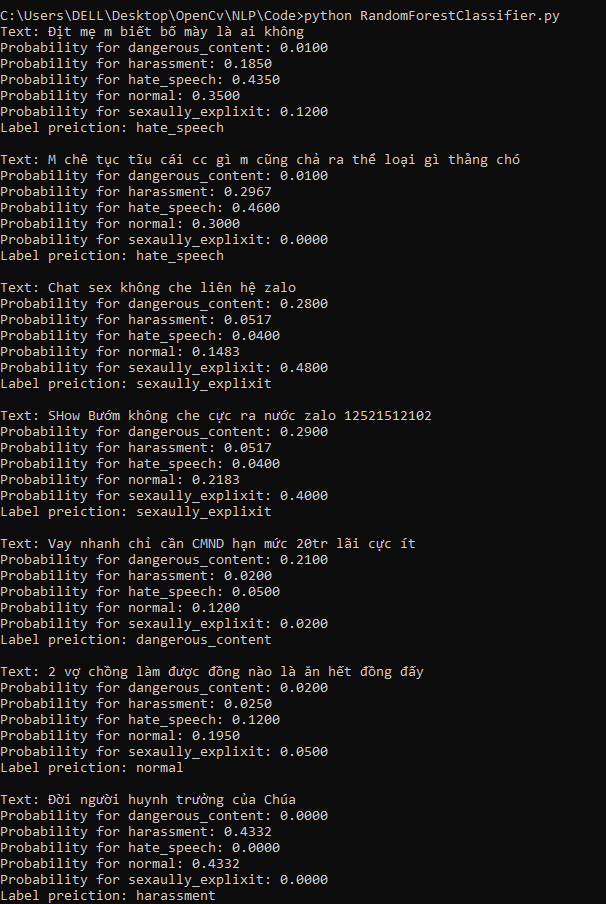
* Decision Trees: It creates multiple decision trees, each trained on a random subset of the data.
* Bagging: Random Forest combines the predictions of these decision trees to make a final prediction, typically by averaging or voting.
* Ensemble Averaging: By averaging the predictions of multiple trees, Random Forest reduces overfitting and improves accuracy.

B/ Implementation:

The script employs the Random Forest classifier for text classification:

* Data Preprocessing: Similar to other script, the data is cleaned and split for training and testing.
* Model Training: Random Forest is trained on the prepared data, learning to classify text.
* Testing and Evaluation: The trained Random Forest model predicts probabilities for each label in the test data, and the predictions are evaluated.

C/ Result:



5. Support Vector Machine (SVM) Model:

A/Introduction:

Support Vector Machine is a powerful classifier that finds the hyperplane that best separates classes in the feature space:

* Maximum Margin: SVM aims to find the hyperplane with the maximum margin between classes.
* Kernel Trick: It can efficiently handle non-linear classification tasks by mapping input features into a higher-dimensional space.
* Classification: Based on the position of input data points relative to the hyperplane, SVM predicts the class for new inputs.

B/Implementation:

The script utilizes Support Vector Machine for text classification:

* Data Preprocessing: The script prepares the data for training by cleaning and splitting it.
* Model Training: SVM is trained on the prepared data, learning to classify text based on TF-IDF values.
* Testing and Evaluation: The trained SVM model predicts labels for the test data, and the predictions are evaluated.

C/Result:  
  
