**VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE & ENGINEERING**

Logo

Description automatically generated

**INTRODUCTION TO ARTIFICIAL INTELLIGENCE**

Course by Dr. Nguyen Trung Ky

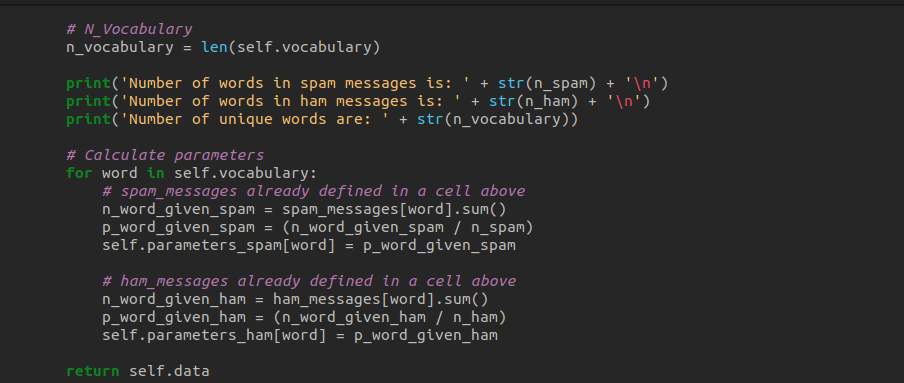
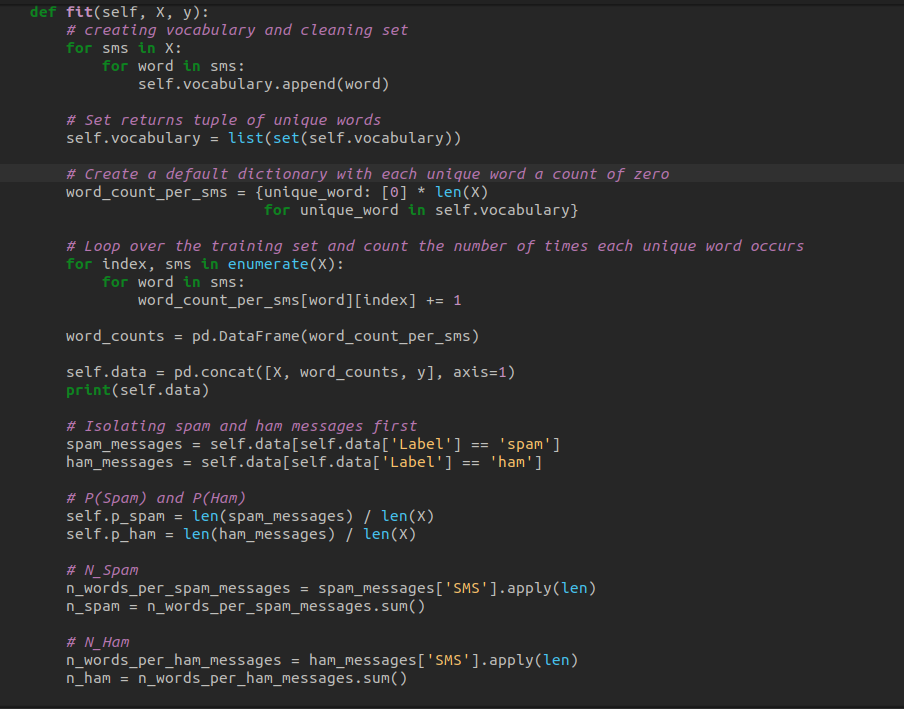
**LAB #5:**

**NAÏVE BAYES (SPAM FILTER)**

Student's Name: Nguyen Duc Nguyen Phuc

Student's ID: ITDSIU21108

**Problem 1:**



Description:

Using this method, a set of unique words is formed by iterating over all the words in the messages in X and then applying the set() function. Every distinct word is used as a key and a list of zeros is used as the matching value to form a dictionary called word\_count\_per\_sms.

After counting the number of times each word appears in each message in X, the code loops over each message and changes the matching count in word\_count\_per\_sms. After converting to a pandas DataFrame word\_counts, the output dictionary is concatenated with the original X and Y data to produce a new DataFrame self.data.

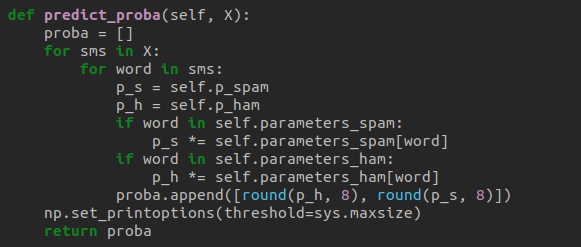
Next, boolean indexing is used to separate the spam and ham messages from self.data. Based on the relative proportions of spam and ham messages, the probability of a message being spam or ham (self.p\_spam and self.p\_ham) are computed.

Together with the total number of unique terms in the vocabulary (n\_vocabulary), the total number of words in the spam and ham messages (n\_spam and n\_ham) are also computed.

Lastly, using the counts of each word in the spam and ham messages, the conditional probabilities of each word in the vocabulary given spam or ham messages (self.parameters\_spam and self.parameters\_ham) are computed.

The self.data DataFrame is returned by the function.

**Problem 2:**



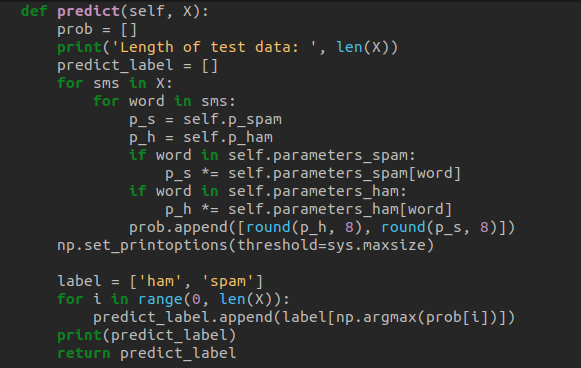
*Description:*

Using the Naive Bayes algorithm, the method iterates over every word in each message in X and determines the message's likelihood of being spam or ham. The prior probabilities self.p\_spam and self.p\_ham are multiplied by the probabilities, and if a word in the appropriate dictionary is present, the conditional probabilities for each word in the message (self.parameters\_spam and self.parameters\_ham) are applied.

The likelihood that the message is spam and the likelihood that it is ham are the two aspects of the probabilities that are added to the probability list.

Lastly, the procedure returns the proba list and sets NumPy's print settings to display the complete array when printed. It should be noted that before being added to probability, the probabilities are rounded to eight decimal places.

**Problem 3:**



Description:

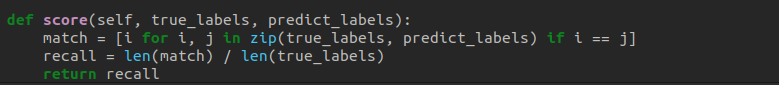
The probability estimates for each message are first stored in an empty list named prob, which is initialized by the method. Next, it goes over every message in X and goes through every word in each message. Using the parameters that were acquired during training, it determines the conditional probability of each word given the message being "ham" or "spam". It gives the "ham" and "spam" classes a probability of 1/2 if the word is absent from the training set. It then calculates the overall chance that the message is "spam" or "ham" by multiplying the conditional probabilities for each word.

The procedure keeps track of the probability estimates for every message in the prob list, which is a collection of lists with the probability estimates for each inner list corresponding to a message. All elements will be printed when the prob list is printed thanks to the np.set\_printoptions(threshold=sys.maxsize) line, which sets the print options for NumPy arrays.

Next, the function initializes an empty list named predict\_label, which will hold the predicted labels for each message, and adds the labels "ham" and "spam" to the label list. The index of the maximum probability is then determined by iterating through each probability estimate in prob and using np.argmax. Based on the index of the maximum likelihood, it then adds the appropriate label ("ham" or "spam") to the predict\_label list.

Ultimately, the predict\_label list is printed and returned as the method's output.

**Problem 4:**



Description:

Using a list comprehension, the technique first generates a list called match. Using the zip function, it simultaneously iterates over the true\_labels and predict\_labels lists, determining if each pair of labels is equal (i.e., whether the prediction is valid). The index of that label is added to the match list if they are equivalent.   
The recall score is then determined by dividing the match list's length by the true\_labels list's length in the technique. The percentage of real positive cases that the model properly classified as positive is known as recall.   
The recall score is finally returned by the technique.