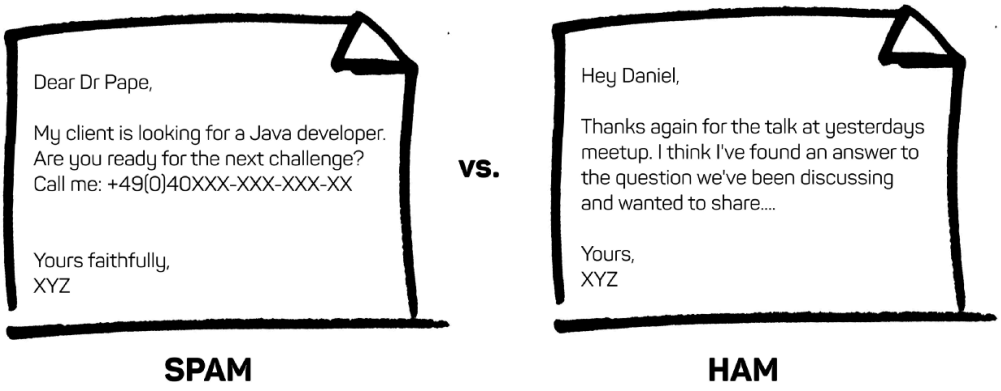
CS471: Introduction to Artificial Intelligence
Naive Bayes
Malcolm Zartman
09/19/2025

Question 1: Naive Bayes Classification

Goal is to classify a given message either a "spam" or a "ham", using a Naive Bayes classifier.



For this assignment, you will be working with a Spam Collection dataset, consisting of text messages that have been collected for Spam research.

Dataset can be downloaded from here: https://drive.google.com/file/d/1MJzF39zlYnTkH4SfvW0TcTVXTrTAeDP-/view?usp=sharing

The csv file contains one message per line with a total of 30 messages tagged either being ham (legitimate) or spam. Each line is composed of two columns: column 1 contains the label (ham or spam) and column 2 contains raw text.

Consider the first 20 samples as your training set and the rest 10 samples for your testing.

Tasks:

1. Load the dataset and split into training and testing sets (first 20 into training and the rest into testing) (1 point)

2. Compute the prior probabilities: P(spam) and P(ham) (2 points)

3. Compute the conditional probabilities P(sentence/spam) (2 points)

4. Compute the posterior probabilities (probability of a sentence belonging to a spam or ham) (2 points)

   P(spam/sentence) $\propto$ P(spam) * P(sentence/spam)

   Posterior $\propto$ prior * conditional

   P(ham/sentence) $\propto$ P(ham) * P(sentence/ham)

5. For each sentence in the test set: (2 points)
   Display the sentence
   Print the posterior probability of a sentence belonging to spam or ham
   Display the class (spam or ham)

6. Report the test set accuracy (1 point)
   Accuracy = no. of sentences correctly predicted by model / total sentences

---

Colab link:
https://colab.research.google.com/drive/1I3ei8mIDZtnpU74m0PYnMYKYvLh-6roX?usp=sharing

1. Load the dataset and split into training and testing sets (first 20 into training and the rest into testing) (1 point)

```
from google.colab import drive
drive.mount('/content/drive/')
file_path = '/content/drive/MyDrive/CS471/SpamDetection.csv'
# loading the dataset
import pandas as pd
df = pd.read_csv(file_path)
df
```

Loaded the dataset using pd.read_csv

```
# splitting into training (first 20) and testing (last 10) sets
training_df = df[:20]
testing_df = df[20:]
```

The first 20 rows of the dataset were put into the training data frame, while the rest of the dataset (last 10 rows) were put into the testing data frame.

2. Compute the prior probabilities: P(spam) and P(ham) (2 points)

This gets the total number of spam and ham values in the training dataset.

```
spam_count = training_df.loc[df['Target'] == 'spam'].shape[0] #
shape gives number of rows
ham_count = training_df.loc[df['Target'] == 'ham'].shape[0]
```

This calculates the prior probabilities by dividing the total number of said classes (spam or ham sentences) by the total number of rows (sentences) in the training dataset.

```
p_spam = spam_count / training_df.shape[0]
p_ham = ham_count / training_df.shape[0] # same as, 1 - P(spam)
```

3. Compute the conditional probabilities P(sentence/spam) (2 points)

The below code I am separating the training data into two separate dataframes (a spam and ham data frame), this is in order to perform calculations more easily.

```
from collections import Counter
import re
```

```
# Compute the conditional probabilities - P(sen|spam) =
P(spam|sen)P(sen)
spam_sentences = training_df.loc[df['Target'] == 'spam']
ham_sentences = training_df.loc[df['Target'] == 'ham']

def tokenizer(sentence):
  # convert to lowercase and split by non-alphanumeric characters
  return re.findall(r'\b\w+\b', sentence.lower())
```

Also, I made a tokenizer function in order to normalize the words by lowercasing them and splitting any non-alphanumeric characters so I leave out excess with actual words. I use the library ‘re’ in order to do that.

Here, I get all of the words from the spam and ham datasets and only get the words from the datasets and normalize it by using my tokenizer function.

```
spam_words = [word for sentence in spam_sentences['data'] for word
in tokenizer(sentence)]
ham_words = [word for sentence in ham_sentences['data'] for word in
tokenizer(sentence)]
```

Here I am setting up other data points I need in order to calculate the conditional probability. This includes me getting the total amount of unique words, counting the number of times a word shows up in each dataset by using the Counter library, and then filling in the rest of the words for the spam and ham datasets and giving the words a 0 value if they don’t show up at all.

```
all_words = spam_words + ham_words
vocab = list(set(all_words))
vocab_size = len(vocab)
print("Count of unique words: ", vocab_size)

# stores total word count for each word
spam_words_count = Counter(spam_words)
ham_words_count = Counter(ham_words)

# make any word not present equal to 0
for word in vocab:
  if word not in spam_words_count:
    spam_words_count[word] = 0
  if word not in ham_words_count:
    ham_words_count[word] = 0
```

```
# get the total number of words in each class
total_spam_words = sum(spam_words_count.values())
total_ham_words = sum(ham_words_count.values())

print("Count of spam words: ", total_spam_words)
print("Count of ham words: ", total_ham_words)
print("Sum should equal to total # of words of: ", len(all_words))
```

Here, I calculate the probabilities (for spam and ham) and perform Laplice smoothing in order to avoid any zeros. I do this by adding one to the numerator and adding the total number of unique words to the denominator.

```
import math

# Dictionaries to store the conditional probabilities
prob_word_given_spam = {}
prob_word_given_ham = {}

# Laplace smoothing constant
alpha = 1

# calculate probabilities for spam
for word in vocab:
    count = spam_words_count[word]
    prob_word_given_spam[word] = (count + alpha) / (total_spam_words
+ vocab_size)

# calculate probabilities for ham
for word in vocab:
    count = ham_words_count[word]
    prob_word_given_ham[word] = (count + alpha) / (total_ham_words +
vocab_size)
```

4. Compute the posterior probabilities (probability of a sentence belonging to a spam or ham) (2 points)

This function takes a sentence, normalizes it, then calculates its posterior probabilities by getting the product of conditional probabilities with prior probabilities of each word and then multiplying them altogether to get the result.

```
# Function that calculates the posterior probability Posterior ∝
prior * conditional given a class.
def calculate_posterior(sentence, prior_prob, cond_prob):
    """
    Parameters:
        sentence (str): The input sentence to classify.
        prior_prob (float): The prior probability of the class
(P(spam) or P(ham)).
        cond_prob (dict): The dictionary of conditional
probabilities for the class.

    Returns the posterior probability.
    """
    # tokenize the sentence and iterate through each word
    tokenized_sentence = tokenizer(sentence)
    posterior = 1 # used to hold the posterior probability
    for word in tokenized_sentence:
        if word in vocab:
            posterior *= prior_prob * cond_prob[word]
    return posterior
```

This function is for testing purposes. It takes in a sentence and then calculates the posterior probability for both spam and ham sentences. After that it compares the probabilities and whichever one is greater it returns that class (‘spam’ or ‘ham’).

```
def spam_or_ham(test_sentence):
  # Compute the posterior probability for spam and ham
  post_spam = calculate_posterior(test_sentence, p_spam,
prob_word_given_spam)
  post_ham = calculate_posterior(test_sentence, p_ham,
prob_word_given_ham)

  print(f"Posterior for Spam: {post_spam:.3e}")
  print(f"Posterior for Ham: {post_ham:.3e}")

  # Compare and classify
  if post_spam > post_ham:
      return 'spam'
```

```
  else:
      return 'ham'
```

5. For each sentence in the test set: (2 points)
   Display the sentence
   Print the posterior probability of a sentence belonging to spam or ham
   Display the class (spam or ham)

   This code iterates through each row of the testing dataset and takes the sentence and plugs it into the function I defined above in order to see if it is ham or spam. If the prediction is right, I will add one to the correct_predictions variable.

```
# TESTING:
#################################################################
###########

correct_predictions = 0 # for seeing accuracy/variance of model
for index, row in testing_df.iterrows():
  sentence = row['data']
  actual_class = row['Target']

  result = spam_or_ham(sentence)

  if result == actual_class:
    correct_predictions += 1

  print(f"Sentence: {sentence}")
  print(f"Predicted: {result}, Actual: {actual_class}")
  print("-" * 50)
```

Results:

```
Posterior for Spam: 1.157e-05
Posterior for Ham: 8.333e-05
Sentence: Tell where you reached
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 7.713e-25
Posterior for Ham: 2.943e-24
Sentence: Your gonna have to pick up a burger for yourself on your way
home
Predicted: ham, Actual: ham
```

```
--------------------------------------------------
Posterior for Spam: 2.296e-36
Posterior for Ham: 5.247e-35
Sentence: As a valued customer I am pleased to advise you that for your
recent review you are awarded a Bonus Prize
Predicted: ham, Actual: spam
--------------------------------------------------
Posterior for Spam: 2.413e-18
Posterior for Ham: 1.413e-19
Sentence: Urgent you are awarded a complimentary trip to EuroDisinc To
claim text immediately
Predicted: spam, Actual: spam
--------------------------------------------------
Posterior for Spam: 2.315e-05
Posterior for Ham: 8.333e-05
Sentence: Finished class where are you
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 3.721e-13
Posterior for Ham: 1.268e-10
Sentence: where are you how did you perform
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 6.512e-13
Posterior for Ham: 3.962e-12
Sentence: you can call me now
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 4.187e-18
Posterior for Ham: 4.952e-16
Sentence: I am waiting Call me once you are free
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 3.979e-13
Posterior for Ham: 1.189e-11
Sentence: I am on the way to homei
Predicted: ham, Actual: ham
--------------------------------------------------
Posterior for Spam: 1.010e-26
Posterior for Ham: 8.396e-28
Sentence: Please call our customer service representative between 10am-9pm
as you have WON a guaranteed cash prize
Predicted: spam, Actual: spam
--------------------------------------------------
```

6. Report the test set accuracy (1 point)

This code evaluates the accuracy of the algorithm by dividing the number of correct predictions (which was tracked when iterating through the sentences) by the total number of sentences.

```
total_sentences = len(testing_df)
accuracy = correct_predictions / total_sentences

print(f"\nModel Accuracy on Test Set: {accuracy * 100:.2f}%")
print(f"Total Correct Predictions: {correct_predictions}")
print(f"Total Sentences: {total_sentences}")
```

Result:

```
Model Accuracy on Test Set: 90.00%
Total Correct Predictions: 9
Total Sentences: 10
```