# Homework 5 - Mathetmatical Foundations of Machine Learning Engineers

kipngeno koech - bkoech

November 24, 2024

## 1 Probability (70 points)

### 1.1 MAP (30 Points)

In this question, we will use the Geometric distribution (the distribution of independent Bernoulli trials until the first success) to model an airport baggage claim. Let $B$ be the number of your bag (meaning you watched $B-1$ bags pass before yours arrived). The probability distribution of $B$ is given by

$$Pr(B=b) = (1-p)^{b-1}p \quad \text{for } b = 1, 2, \ldots$$

where $0 < p \leq 1$ is the parameter of the distribution.

We now will use a Beta prior to get a MAP estimate of the distribution parameter $p$. Specifically,

$$Pr(p|\alpha, \beta) = p^{\alpha-1}(1-p)^{\beta-1}$$

where $\alpha > 0$ and $\beta > 0$ are fixed, given constants.

1. Assume that over $n$ trips, you noted your bag's number as $b_1, b_2, \ldots, b_n$ respectively, and the number of your bag on each of these trips is independent. Derive the log posterior probability of recording these numbers $b_1, b_2, \ldots, b_n$:

$$\log Pr(p|b_1, \ldots, b_n)$$

(10 Points)

*Hint:* you can use Bayes' Rule.
Baye's rule: Baye's rule in this case is the posterior probability of $p$ given $b_1, \ldots, b_n$ is given by:
geometric distribution:

$$Pr(B=b) = (1-p)^{b-1}p$$

beta prior:

$$Pr(p|\alpha, \beta) = p^{\alpha-1}(1-p)^{\beta-1}$$

Bayes theorem:

$$Pr(A|B) = \frac{Pr(B|A)Pr(A)}{Pr(B)}$$

where $A$ and $B$ in this case are the parameters $p$ and $b_1, \ldots, b_n$ respectively. Therefore, the posterior probability of $p$ given $b_1, \ldots, b_n$ is:

$$Pr(p|b_1, \ldots, b_n) = \frac{Pr(b_1, \ldots, b_n|p)Pr(p)}{Pr(b_1, \ldots, b_n)}$$

here $Pr(b_1, \ldots, b_n)$ is the marginal probability of $b_1, \ldots, b_n$ which can be calculated as:

$$Pr(b_1, \ldots, b_n) = \int Pr(b_1, \ldots, b_n|p)Pr(p)dp$$

where $Pr(b_1, \ldots, b_n|p)$ is the likelihood of $b_1, \ldots, b_n$ given $p$ and $Pr(p)$ is the prior probability of $p$. Therefore, the log posterior probability of recording these numbers $b_1, b_2, \ldots, b_n$ is:

$$\log Pr(p|b_1, \ldots, b_n) = \log\left(\frac{Pr(b_1, \ldots, b_n|p)Pr(p)}{Pr(b_1, \ldots, b_n)}\right)$$

since the log of a fraction is the difference of the logs of the numerator and the denominator:

$$= \log Pr(b_1, \ldots, b_n|p) + \log Pr(p) - \log Pr(b_1, \ldots, b_n)$$

but the $Pr(b_1, \ldots, b_n)$ does not depend on $p$ so we can ignore it:

$$= \log Pr(b_1, \ldots, b_n|p) + \log Pr(p) + C$$

where $C$ is a constant (to represent what doesn't depend on p)
we find the product of $Pr(b_1, \ldots, b_n)$ because it is a product of independent events:

$$= \log\left(\prod_{i=1}^{n}(1-p)^{b_i-1}p\right) + \log\left(p^{\alpha-1}(1-p)^{\beta-1}\right) + C$$

so our log posterior probability of recording these numbers $b_1, b_2, \ldots, b_n$ is:

$$= \sum_{i=1}^{n}\log\left((1-p)^{b_i-1}p\right) + \log\left(p^{\alpha-1}(1-p)^{\beta-1}\right) + C$$

if we expand this further:

$$= \sum_{i=1}^{n}\left((b_i-1)\log(1-p) + \log p\right) + (\alpha-1)\log p + (\beta-1)\log(1-p) + C$$

2. Use your answer in part (1) to derive the maximum-a-posterior estimate (MAP) of the parameter:

$$\hat{p} = \arg\max_{0<p\leq 1} \log Pr(p|b_1, \ldots, b_n)$$

(10 Points)

$$\hat{p} = \arg\max_{0<p\leq 1} \log Pr(p|b_1, \ldots, b_n)$$

our log posterior probability of recording these numbers $b_1, b_2, \ldots, b_n$ is:

$$= \sum_{i=1}^{n} ((b_i - 1)\log(1-p) + \log p) + (\alpha - 1)\log p + (\beta - 1)\log(1-p) + C$$

to find the maximum of this function, we take the derivative with respect to $p$ and set it to zero:

$$\frac{d}{dp}\log Pr(p|b_1, \ldots, b_n) = \frac{d}{dp}\left(\sum_{i=1}^{n} ((b_i - 1)\log(1-p) + \log p) + (\alpha - 1)\log p + (\beta - 1)\log(1-p) + C\right)$$

$$= \sum_{i=1}^{n}\left(\frac{d}{dp}(b_i - 1)\log(1-p) + \frac{d}{dp}\log p\right) + \frac{d}{dp}(\alpha - 1)\log p + \frac{d}{dp}(\beta - 1)\log(1-p) + \frac{d}{dp}C$$

$$= \sum_{i=1}^{n}\left(\frac{d}{dp}(b_i - 1)\log(1-p) + \frac{d}{dp}\log p\right) + \frac{d}{dp}(\alpha - 1)\log p + \frac{d}{dp}(\beta - 1)\log(1-p) + 0$$

$$= \sum_{i=1}^{n}\left(\frac{d}{dp}(b_i - 1)\log(1-p) + \frac{d}{dp}\log p\right) + \frac{\alpha - 1}{p} - \frac{\beta - 1}{1-p}$$

the derivatives inside the summation are:

$$\frac{d}{dp}(b_i - 1)\log(1-p) = (b_i - 1)\frac{d}{dp}\log(1-p) = (b_i - 1)\frac{-1}{1-p} = \frac{1-b_i}{1-p}$$

$$\frac{d}{dp}\log p = \frac{1}{p}$$

so the derivative of the log posterior probability of recording these numbers $b_1, b_2, \ldots, b_n$ is:

$$= \sum_{i=1}^{n}\left(-\frac{b_i - 1}{1-p} + \frac{1}{p}\right) + \frac{\alpha - 1}{p} - \frac{\beta - 1}{1-p}$$

simplifying this:

$$= \sum_{i=1}^{n}\left(\frac{1}{p} - \frac{b_i - 1}{1-p}\right) + \frac{\alpha - 1}{p} - \frac{\beta - 1}{1-p}$$

setting this to zero to find the maximum:

$$\sum_{i=1}^{n}\left(\frac{1}{p} - \frac{b_i - 1}{1-p}\right) + \frac{\alpha - 1}{p} - \frac{\beta - 1}{1-p} = 0$$

simplifying this:

$$\frac{\alpha - 1 + n}{p} - \frac{\beta - 1 + \sum_{i=1}^{n}(b_i - 1)}{1-p} = 0$$

take the first term to the other side:

$$\frac{\alpha - 1 + n}{p} = \frac{\beta - 1 + \sum_{i=1}^{n}(b_i - 1)}{1-p}$$

cross multiply:

$$(\alpha - 1 + n)(1-p) = p(\beta - 1 + \sum_{i=1}^{n}(b_i - 1))$$

expand the left side:

$$\alpha - 1 + n - p(\alpha - 1 + n) = p(\beta - 1 + \sum_{i=1}^{n}(b_i - 1))$$

take the $p$ terms to the righ side:

$$\alpha - 1 + n = p(\alpha - 1 + n + \beta - 1 + \sum_{i=1}^{n}(b_i - 1))$$

so:

$$\hat{p} = \frac{\alpha - 1 + n}{\alpha - 2 + n + \beta + \sum_{i=1}^{n}(b_i - 1)}$$

3. Suppose that $n = 5$ and the values of $b_1, b_2, \ldots, b_n$ are 10, 9, 5, 28, 7. Also, $\alpha = 14$ and $\beta = 590$. Using your answer in part (2), what is the MAP estimate of the $p$ parameter for this data and prior? (10 Points)

$$\hat{p} = \frac{\alpha - 1 + n}{\alpha - 2 + n + \beta + \sum_{i=1}^{n}(b_i - 1)}$$

$$\hat{p} = \frac{14 - 1 + 5}{14 - 2 + 5 + 590 + 10 + 9 + 5 + 28 + 7 - 5}$$

$$\hat{p} = \frac{18}{14 + 5 + 590 + 10 + 9 + 5 + 28 + 7 - 5}$$

$$\hat{p} = \frac{18}{663} = \frac{6}{221} = \mathbf{0.027149}$$

## 1.2 Naive Bayes (20 Points)

You are asked to build a Naïve Bayes classifier using the training dataset in Table 1, where each instance is assigned to one out of 3 classes ("healthy" (H), "influenza" (I), or "salmonella poisoning" (S)).

| Training | Fever (F) | Vomiting (V) | Diarrhea (D) | Classification |
|---|---|---|---|---|
| D1 | no | no | no | Healthy (H) |
| D2 | average | no | no | Influenza (I) |
| D3 | high | no | no | Influenza (I) |
| D4 | high | yes | yes | Salmonella poisoning (S) |
| D5 | average | no | yes | Salmonella poisoning (S) |

Table 1: Health Classification based on Symptoms

1. Using the Naïve Bayes model, find the prior probabilities $P(H)$, $P(I)$, and $P(S)$ given the training data above. (4 points)

$$P(H) = \mathbf{\frac{1}{5}}, \quad P(I) = \frac{2}{5}, \quad P(S) = \frac{2}{5}$$

2. Fill in Table 2 below to finish building your Naïve Bayes classifier. Use Laplace smoothing with parameter $\alpha = 2$ for your conditional probability estimates. (8 points)

$$P(X|Y) = \frac{N_{X,Y} + \alpha}{N_Y + \alpha \cdot |X|}$$

where $N_{X,Y}$ is the number of occurrences of $X$ in class $Y$, $N_Y$ is the number of occurrences of class $Y$, and $|X|$ is the number of possible values of $X$.

$$P(X|Y) = \frac{N_{X,Y} + 2}{N_Y + 2 \cdot |X|}$$

for $X$ and $Y$ in the table below.
$x$ = high fever:

$$P(x|H) = \frac{0+2}{1+2\cdot 3} = \mathbf{\frac{2}{7}}$$
$$P(x|I) = \frac{1+2}{2+2\cdot 3} = \mathbf{\frac{3}{8}}$$
$$P(x|S) = \frac{1+2}{2+2\cdot 3} = \mathbf{\frac{3}{8}}$$

$x$ = average fever:

$$P(x|H) = \frac{0+2}{1+2\cdot 3} = \mathbf{\frac{2}{7}}$$
$$P(x|I) = \frac{1+2}{2+2\cdot 3} = \mathbf{\frac{3}{8}}$$
$$P(x|S) = \frac{1+2}{2+2\cdot 3} = \mathbf{\frac{3}{8}}$$

$x$ = fever no:

$$P(x|H) = \frac{1+2}{1+2\cdot 3} = \mathbf{\frac{3}{7}}$$
$$P(x|I) = \frac{0+2}{2+2\cdot 3} = \mathbf{\frac{1}{4}}$$
$$P(x|S) = \frac{0+2}{2+2\cdot 3} = \mathbf{\frac{1}{4}}$$

$x$ = vomiting yes:

$$P(x|H) = \frac{0+2}{1+2\cdot 2} = \mathbf{\frac{2}{5}}$$
$$P(x|I) = \frac{0+2}{2+2\cdot 2} = \mathbf{\frac{2}{6}}$$
$$P(x|S) = \frac{1+2}{2+2\cdot 2} = \mathbf{\frac{3}{6}}$$

$x$ = vomiting no:

$$P(x|H) = \frac{1+2}{1+2\cdot 2} = \mathbf{\frac{3}{5}}$$
$$P(x|I) = \frac{2+2}{2+2\cdot 2} = \mathbf{\frac{4}{6}}$$
$$P(x|S) = \frac{1+2}{2+2\cdot 2} = \mathbf{\frac{3}{6}}$$

$x$ = diarrhea = yes:

$$P(x|H) = \frac{0+2}{1+2\cdot 2} = \mathbf{\frac{2}{5}}$$
$$P(x|I) = \frac{0+2}{2+2\cdot 2} = \mathbf{\frac{2}{6}}$$
$$P(x|S) = \frac{2+2}{2+2\cdot 2} = \mathbf{\frac{4}{6}}$$

$x$ = diarrhea = no:

$$P(x|H) = \frac{1+2}{1+2\cdot 2} = \mathbf{\frac{3}{5}}$$
$$P(x|I) = \frac{2+2}{2+2\cdot 2} = \mathbf{\frac{4}{6}}$$
$$P(x|S) = \frac{0+2}{2+2\cdot 2} = \mathbf{\frac{2}{6}}$$

| $P(X\|Y)$ | $Y = H$ | $Y = I$ | $Y = S$ |
|---|---|---|---|
| $X =$ (high fever) | $\frac{2}{7}$ | $\frac{3}{8}$ | $\frac{3}{8}$ |
| $X =$ (average fever) | $\frac{2}{7}$ | $\frac{3}{8}$ | $\frac{3}{8}$ |
| $X =$ (no fever) | $\frac{3}{7}$ | $\frac{1}{4}$ | $\frac{1}{4}$ |
| $X = V = Yes$ | $\frac{2}{5}$ | $\frac{2}{6}$ | $\frac{3}{6}$ |
| $X = V = No$ | $\frac{3}{5}$ | $\frac{4}{6}$ | $\frac{3}{6}$ |
| $X = D = yes$ | $\frac{2}{5}$ | $\frac{2}{6}$ | $\frac{4}{6}$ |
| $X = D = no$ | $\frac{3}{5}$ | $\frac{4}{6}$ | $\frac{2}{6}$ |

Table 2: Conditional Probability Table for $P(X|Y)$

3. Apply your Naïve Bayes Classifier to a person who is vomiting but has no fever or diarrhea. Determine the probabilities of this person being healthy, suffering from influenza, and salmonella poisoning. (8 points)
Let $V$ be vomiting, $F$ be fever, and $D$ be diarrhea. The probabilities of this person being healthy, suffering from influenza, and salmonella poisoning are:
Naive Bayes classifier:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

where $X$ is the evidence and $Y$ is the class. We can ignore the denominator since it is constant for all classes:

$$P(Y|X) \propto P(X|Y)P(Y)$$

probability that the person is healthy:

$$P(H|V = \text{yes}, F = \text{no}, D = \text{no}) \propto P(V = \text{yes}|H)P(F = \text{no}|H)P(D = \text{no}|H)P(H)$$
$$= \frac{2}{5}\cdot\frac{3}{7}\cdot\frac{3}{5}\cdot\frac{1}{5} = \frac{18}{875}$$

probability that the person is suffering from influenza:

$$P(I|V = \text{yes}, F = \text{no}, D = \text{no}) \propto P(V = \text{yes}|I)P(F = \text{no}|I)P(D = \text{no}|I)P(I)$$
$$= \frac{2}{6}\cdot\frac{1}{4}\cdot\frac{4}{6}\cdot\frac{2}{5} = \frac{1}{45}$$

probability that the person is suffering from salmonella poisoning:

$$P(S|V = \text{yes}, F = \text{no}, D = \text{no}) \propto P(V = \text{yes}|S)P(F = \text{no}|S)P(D = \text{no}|S)P(S)$$
$$= \frac{3}{6}\cdot\frac{1}{4}\cdot\frac{2}{6}\cdot\frac{2}{5} = \frac{1}{60}$$

let us sum the probabilities:

$$P(H|V = \text{yes}, F = \text{no}, D = \text{no}) + P(I|V = \text{yes}, F = \text{no}, D = \text{no}) + P(S|V = \text{yes}, F = \text{no}, D = \text{no}) = \frac{18}{875} + \frac{1}{45} + \frac{1}{60} = \frac{1873}{31500}$$

Normalizing the probabilities:

$$P(H|V = \text{yes}, F = \text{no}, D = \text{no}) = \frac{18}{875}\cdot\frac{31500}{1873} = \frac{36}{1873} = \mathbf{0.345969}$$
$$P(I|V = \text{yes}, F = \text{no}, D = \text{no}) = \frac{1}{45}\cdot\frac{31500}{1873} = \frac{700}{1873} = \mathbf{0.373978}$$
$$P(S|V = \text{yes}, F = \text{no}, D = \text{no}) = \frac{1}{60}\cdot\frac{31500}{1873} = \frac{525}{1873} = \mathbf{0.280053}$$

The probabilities of this person being healthy, suffering from influenza, and salmonella poisoning are 0.345969, 0.373978, and 0.280053 respectively. This shows that this person is more likely to be suffering from influenza.

## 1.3 MLE (20 points)

- A variable $y$ is called a count if it only takes non-negative integer values, i.e., $y \in \{0, 1, 2, \ldots\}$. A common distribution for handling such variables is the Poisson distribution, which has the following form:

$$\text{Poisson}(y; \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}, \quad y \in \{0, 1, 2, \ldots\}$$

  where $\lambda > 0$ is some known constant that characterizes the Poisson distribution.

  Given independent identically distributed (iid) observations $\{y_i\}_{i=1}^N$, $y_i \in \{0, 1, 2, \ldots\}$, calculate the MLE estimate of $\lambda$. (8 Points)

$$L_Y(\lambda) = \prod_{i=1}^N \frac{e^{-\lambda}\lambda^{y_i}}{y_i!} = \frac{e^{-N\lambda}\lambda^{\sum_{i=1}^N y_i}}{\prod_{i=1}^N y_i!}$$

$$\log L_Y(\lambda) = -N\lambda + \sum_{i=1}^N y_i \log\lambda - \sum_{i=1}^N \log y_i!$$

$$\frac{d}{d\lambda}\log L_Y(\lambda) = -N + \frac{1}{\lambda}\sum_{i=1}^N y_i$$

$$\frac{d}{d\lambda}\log L_Y(\lambda) = 0 \implies \lambda = \frac{1}{N}\sum_{i=1}^N y_i$$

$$\hat{\lambda}_{\text{MLE}} = \frac{1}{N}\sum_{i=1}^N y_i$$

- In this problem, you will examine the task of estimating the probability density of the maximum height obtained by waves in the ocean. Scientists have recorded the maximum wave height on $n$ days, obtaining samples $x_1, x_2, \ldots, x_n \in \mathbb{R}$. It is known that these are i.i.d. random variables following the Rayleigh distribution with parameter $\theta$. Consider the following probability density function for the Rayleigh distribution:

$$f_X(x; \theta) = \frac{x}{\theta^2}\exp\left(-\frac{x^2}{2\theta^2}\right)$$

  The likelihood function for your estimate is then $L_X(\theta) = f_X(x_1, \ldots, x_n|\theta)$. Your task is to estimate $\hat{\theta}_{\text{MLE}}$, the maximum likelihood estimate of $\theta$. (12 Points)

  The likelihood function is:

$$L_X(\theta) = \prod_{i=1}^n \frac{x_i}{\theta^2}\exp\left(-\frac{x_i^2}{2\theta^2}\right) = \frac{1}{\theta^{2n}}\exp\left(-\frac{1}{2\theta^2}\sum_{i=1}^n x_i^2\right)\prod_{i=1}^n x_i$$

$$\log L_X(\theta) = -2n\log\theta - \frac{1}{2\theta^2}\sum_{i=1}^n x_i^2 + \sum_{i=1}^n \log x_i$$

  dropping the values that do not depend on $\theta$ since their derivative will be zero:

$$\log L_X(\theta) = -2n\log\theta - \frac{1}{2\theta^2}\sum_{i=1}^n x_i^2$$

$$\frac{d}{d\theta}\log L_X(\theta) = -\frac{2n}{\theta} + \frac{1}{\theta^3}\sum_{i=1}^n x_i^2$$

  we then set the derivative to zero to find the maximum likelihood estimate of $\theta$:

$$-\frac{2n}{\theta} + \frac{1}{\theta^3}\sum_{i=1}^n x_i^2 = 0$$

$$\frac{2n}{\theta} = \frac{1}{\theta^3}\sum_{i=1}^n x_i^2$$

  multiply both sides by $\theta^3$:

$$2n\theta^2 = \sum_{i=1}^n x_i^2$$

  divide both sides by $2n$:

$$\hat{\theta}_{\text{MLE}} = \sqrt{\frac{1}{2n}\sum_{i=1}^n x_i^2}$$

## 2 Text classification (30 Points)

Consider a text classification problem. In this case, you will try to classify text as either spam or ham. To do this, you will apply concepts of Likelihood, prior, and posterior given a dataset comprising pairs of text and labels. There are two types of labels: 1 (spam) and 0 (ham). Your goal is to create a simple classifier that, when given, determines if the text is spam or ham. You have been provided with the starter code and the data

1. Find the priors. What are the priors in this distribution? i.e find $P(ham)$ and $P(spam)$
2. Find the likelihoods for each word. For each word in the dataset, find the likelihood that the word is in spam and ham. This will represent the conditional probability $P(w|\text{spam})$ and $P(w|\text{ham})$ for $w$ where $w \in V$. $V$ is the vocabulary of the dataset.
3. Define a function that, when given a text sequence, returns the probability of the text being in spam. I.e., it returns $P(\text{spam}|\text{text})$. Note that this function calculates the likelihood using the Bayes rule. Do the same for ham.
4. Perform inference, i.e., given a string of text, determine if it is ham or spam based on the posterior probabilities calculated from the previous steps. Your function will determine the posterior probability of your text being in ham and spam and classify it as being the larger of the two.
5. Evaluate the data based on your test set and report the accuracy of your classifier. Your accuracy must be greater than 85%.

# Homework5

November 24, 2024

## 1 Spam ham classifier

Consider a text classification problem. In this case, you will try to classify text as either spam or ham. To do this, you will apply concepts of Likelihood, prior, and posterior given a dataset comprising pairs of text and labels. There are two types of labels: 1 (spam) and 0 (ham). Your goal is to create a simple classifier that, when given, determines if the text is spam or ham.

[30]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords
from sklearn.metrics import accuracy_score
import re
```

### 1.1 Data loading and cleaning

[31]:
```python
data = pd.read_csv('spam_ham_dataset.csv')
data = data.dropna()
data = data.drop_duplicates()
df =data
nltk.download('stopwords')
stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /home/kip/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[32]:
```python
def remove_stopwords(text):
    text = [word.lower() for word in text.split() if word.lower() not in stop]
    # remove special characters
    text = [re.sub('\W+','', word) for word in text]

    return " ".join(text)

txt1 = df.text[0]
```

```
print(txt1)
# remove the filler words
txt1 = remove_stopwords(txt1)
print(txt1)
# apply the function to the entire dataset
df['text'] = df['text'].apply(remove_stopwords)
df.head()
```

```
<>:4: SyntaxWarning: invalid escape sequence '\W'
<>:4: SyntaxWarning: invalid escape sequence '\W'
/tmp/ipykernel_108095/2336932942.py:4: SyntaxWarning: invalid escape sequence
'\W'
  text = [re.sub('\W+','', word) for word in text]
```

```
Subject: enron methanol ; meter # : 988291
this is a follow up to the note i gave you on monday , 4 / 3 / 00 { preliminary
flow data provided by daren } .
please override pop ' s daily volume { presently zero } to reflect daily
activity you can obtain from gas control .
this change is needed asap for economics purposes .
subject enron methanol  meter   988291 follow note gave monday  4  3  00
preliminary flow data provided daren   please override pop  daily volume
presently zero  reflect daily activity obtain gas control  change needed asap
economics purposes
```

[32]:
```
   Unnamed: 0 label                                               text  \
0         605   ham  subject enron methanol  meter   988291 follow …
1        2349   ham  subject hpl nom january 9  2001  see attached …
2        3624   ham  subject neon retreat ho ho ho  around wonderf…
3        4685  spam  subject photoshop  windows  office  cheap  mai…
4        2030   ham  subject  indian springs deal book teco pvr rev…

   label_num
0          0
1          0
2          0
3          1
4          0
```

Next we split the data into training and testing. We will derive the probabilities from the training data and then use them to predict the testing data.

[33]:
```
# lets split the data into training and testing
X = df.text
y = df.label_num
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[34]:
```
X_train.shape, X_test.shape
```

[34]: ((4136,), (1035,))

[35]:
```
X_train.head()
```

[35]:
```
5132    subject april activity surveys starting collec…
2067    subject message subject hey  julie  _   turned…
4716    subject txu fuels  sds nomination may 2001 att…
4710    subject  richardson volumes nov 99 dec 99 mete…
2268    subject new era online medical care  new era o…
Name: text, dtype: object
```

### 1.1.1 (1)

Find the priors. What are the priors in this distribution? i.e find P (ham) and P (spam)

[36]:
```
# Calculate priors
P_ham = y_train.value_counts()[0] / len(y_train)
P_spam = y_train.value_counts()[1] / len(y_train)

print(f"P(ham): {P_ham}")
print(f"P(spam): {P_spam}")
```

```
P(ham): 0.7084139264990329
P(spam): 0.2915860735009671
```

### 1.1.2 (2)

Find the likelihoods for each word. For each word in the dataset, find the likelihood that the word is in spam and ham. This will represent the conditional probability P (w|spam) and P (w|ham) for w where w   V . V is the vocabulary of the dataset.

[37]:
```
from collections import defaultdict, Counter

# Initialize counters for spam and ham words
spam_words = Counter()
ham_words = Counter()

# Separate spam and ham texts
spam_texts = X_train[y_train == 1]
ham_texts = X_train[y_train == 0]

# Count words in spam and ham texts
for text in spam_texts:
    for word in text.split():
        spam_words[word] += 1

for text in ham_texts:
    for word in text.split():
        ham_words[word] += 1
```

```
# Calculate total number of words in spam and ham texts
total_spam_words = sum(spam_words.values())
total_ham_words = sum(ham_words.values())

# Calculate likelihoods
likelihoods_spam = {word: (count / total_spam_words) for word, count in␣
 ↪spam_words.items()}
likelihoods_ham = {word: (count / total_ham_words) for word, count in ham_words.
 ↪items()}

print("Likelihoods for spam words:", list(likelihoods_spam.items())[:10])
print("Likelihoods for ham words:", list(likelihoods_ham.items())[:10])
```

```
Likelihoods for spam words: [('subject', 0.008392657843131073), ('message',
0.0015133267335473 01), ('hey', 0.00020808242586275387), ('julie',
1.2611056112894174e-05), ('_', 0.004785895794843339), ('turned',
8.827739279025922e-05), ('18', 0.00030266534670946016), ('high',
0.0008134131192816742), ('school', 8.827739279025922e-05), ('senior',
8.197186473381214e-05)]
Likelihoods for ham words: [('subject', 0.016332990630260 14), ('april',
0.0013167581802791138), ('activity', 0.0006810818173857485), ('surveys',
4.216220774292729e-05), ('starting', 0.00016216233747279728), ('collect',
3.8918960993471346e-05), ('data', 0.000535135713660231), ('attached',
0.002737300256540818), ('survey', 0.00037945986968634563), ('drives',
1.6216233747279726e-05)]
```

### 1.1.3 (3)

Define a function that, when given a text sequence, returns the probability of the text being in spam. I.e., it returns P (spam|text). Note that this function calculates the likelihood using the Bayes rule. Do the same for ham.

[38]:
```
def calculate_posterior(text, priors, likelihoods, total_words):
    '''
    Calculate the posterior probability of a text being spam or ham
    given the text and the likelihoods of spam and ham words.

    Parameters:
    text (str): the text to classify
    priors (tuple): the prior probabilities of spam and ham
    likelihoods (dict): the likelihoods of spam and ham words
    total_words (int): the total number of words in the training set

    returns:
    float: the posterior probability of the text being spam or ham
    '''
    # Split the text into words
    words = text.split()
    # Initialize posterior as the log of the priors
    posterior = np.log(priors)

    # Calculate the posterior for spam and ham of the text
    for word in words:
        # If the word is in the likelihoods dictionary, add the log likelihood␣
 ↪to the posterior
        if word in likelihoods:
            posterior += np.log(likelihoods[word])
        #  If the word is not in the likelihoods dictionary, apply Laplace␣
 ↪smoothing to avoid zero probabilities
        else:
            # Apply Laplace smoothing for unseen words
            posterior += np.log(1 / (total_words + len(likelihoods)))

    # Return the final posterior probability
    return posterior

def predict_spam(text):
    '''
    Predict whether a text is spam given the text.

    Parameters:
    text (str): the text to classify

    returns:
    float: the posterior probability of the text being spam
    '''
    P_spam_given_text = calculate_posterior(text, P_spam, likelihoods_spam,␣
 ↪total_spam_words)
    return P_spam_given_text

def predict_ham(text):
    '''
    Predict whether a text is ham given the text.

    Parameters:
    text (str): the text to classify

    returns:
    float: the posterior probability of the text being ham
    '''
    P_ham_given_text = calculate_posterior(text, P_ham, likelihoods_ham,␣
 ↪total_ham_words)
    return P_ham_given_text
```

```
# Example usage
text_example = "Congratulations, spin and win money now"
print(f"P(spam|text): {predict_spam(text_example)}")
print(f"P(ham|text): {predict_ham(text_example)}")
```

```
P(spam|text): -65.67705077597867
P(ham|text): -70.95047223507962
```

### 1.1.4 (4)

Perform inference, i.e., given a string of text, determine if it is ham or spam based on the posterior probabilities calculated from the previous steps. Your function will determine the posterior probability of your text being in ham and spam and classify it as being the larger of the two.

[39]:
```
def classify_text(text):
    P_spam_given_text = predict_spam(text)
    P_ham_given_text = predict_ham(text)

    if P_spam_given_text > P_ham_given_text:
        return 'spam'
    else:
        return 'ham'

# Example usage
text_example = "Congratulations, spin and win money now"
classification = classify_text(text_example)
print(f"The text '{text_example}' is classified as: {classification}")
```

```
The text 'Congratulations, spin and win money now' is classified as: spam
```

### 1.1.5 (5)

Evaluate the data based on your test set and report the accuracy of your classifier. Your accuracy must be greater than 85%.

[40]:
```
# Predict the labels for the test set
y_pred = X_test.apply(classify_text)

# Convert predictions to numerical labels
y_pred_num = y_pred.apply(lambda x: 1 if x == 'spam' else 0)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_num)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 97.00%
```

## 2 LOGISTIC REGRESSION

### 2.1 1. FROM SKLEARN

[41]:
```
# Import necessary libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

# Convert text data to numerical features using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')  # Removing common English␣
 ↪stopwords
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Initialize and train the logistic regression model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if convergence␣
 ↪is not achieved
model.fit(X_train_tfidf, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_tfidf)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 98.94%
```

### 2.2 2. FROM SCRATCH

[ ]:
```
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Logistic regression equation: z = X * weights + bias
# Sigmoid function: sigmoid(z) = 1 / (1 + exp(-z))


class LogisticRegressionScratch:
    def __init__(self, learning_rate=0.01, n_iters=1000, init_method="random",␣
 ↪lambda_=0.1):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
        self.init_method = init_method
        self.lambda_ = lambda_  # Regularization strength
    def initialize_weights(self, n_features):
        if self.init_method == "random":
            self.weights = np.random.normal(0, 0.01, size=n_features)
        elif self.init_method == "xavier":
            limit = np.sqrt(1 / n_features)
            self.weights = np.random.uniform(-limit, limit, size=n_features)
        elif self.init_method == "he":
            limit = np.sqrt(2 / n_features)
            self.weights = np.random.normal(0, limit, size=n_features)
        else:
            self.weights = np.zeros(n_features)

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-np.clip(z, -500, 500)))  # Numerical stability

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.initialize_weights(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            # Linear combination (use sparse matrix's dot method)
            z = X.dot(self.weights) + self.bias
            y_pred = self.sigmoid(z)

            # Gradients with L2 regularization
            dw = (1 / n_samples) * X.T.dot(y_pred - y) + (self.lambda_ /␣
 ↪n_samples) * self.weights  # Regularization term
            db = (1 / n_samples) * np.sum(y_pred - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        z = X.dot(self.weights) + self.bias
        probabilities = self.sigmoid(z)
        return [1 if prob >= 0.5 else 0 for prob in probabilities]


scaler = StandardScaler(with_mean=False)  # Prevent centering for sparse␣
 ↪matrices
X_train_scaled = scaler.fit_transform(X_train_tfidf)
X_test_scaled = scaler.transform(X_test_tfidf)


# Train and evaluate
```

```
regressor = LogisticRegressionScratch(learning_rate=0.001, n_iters=10000,␣
  ↪init_method="xavier", lambda_=0.1)
regressor.fit(X_train_scaled, y_train)
predictions = regressor.predict(X_test_scaled)

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 95.56%
```

[43]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[734   8]
 [ 38 255]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       742
           1       0.97      0.87      0.92       293

    accuracy                           0.96      1035
   macro avg       0.96      0.93      0.94      1035
weighted avg       0.96      0.96      0.95      1035
```