CS464 Introduction to Machine Learning - Homework 1

Kadir İhsan Sayıcı

October 27, 2024

Question 1.1

We are given two boxes with different types of coins. Let's calculate the probability of getting two tails in a row.

$P(\text{TT} \mid B) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$ (Probability of getting two tails given that you have chosen the blue coin)

$$P(\text{TT} \mid Y) = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$$

$$P(\text{TT} \mid R) = \frac{9}{10} \times \frac{9}{10} = \frac{81}{100}$$

Now, let's calculate the probability of picking each coin and getting two tails:

$$P(\text{Box 1, Blue Coin, TT}) = \frac{1}{2} \times \frac{2}{3} \times \frac{1}{4} = \frac{1}{12}$$

$$P(\text{Box 1, Yellow Coin, TT}) = \frac{1}{2} \times \frac{1}{3} \times \frac{9}{16} = \frac{3}{32}$$

$$P(\text{Box 2, Blue Coin, TT}) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{4} = \frac{1}{16}$$

$$P(\text{Box 2, Red Coin, TT}) = \frac{1}{2} \times \frac{1}{2} \times \frac{81}{100} = \frac{81}{400}$$

Therefore, the total probability of getting two tails is:

$$P(\text{Two tails}) = \frac{1}{12} + \frac{3}{32} + \frac{1}{16} + \frac{81}{400} \approx 0.44208$$

Question 1.2

The coin is fair only if it is the Blue one. Using Bayes' Theorem:

$$P(B \mid \mathrm{TT}) = \frac{P(\mathrm{TT} \mid B) \cdot P(B)}{P(\mathrm{TT})}$$

Where:

$$P(\mathrm{TT} \mid B) = \frac{1}{4}$$

$$P(B) = \frac{1}{2} \times \frac{2}{3} + \frac{1}{2} \times \frac{1}{2} = \frac{7}{12}$$

$$P(\mathrm{TT}) = 0.44208 \quad \text{(calculated in Question 1.1)}$$

Therefore,

$$P(B \mid \mathrm{TT}) = \frac{\frac{1}{4} \times \frac{7}{12}}{0.44208} \approx 0.32988$$

Question 1.3

Using Bayes' Theorem to find the probability of selecting the red coin:

$$P(R \mid \mathrm{TT}) = \frac{P(\mathrm{TT} \mid R) \cdot P(R)}{P(\mathrm{TT})}$$

Where:

$$P(\mathrm{TT} \mid R) = \frac{81}{100}$$

$$P(R) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

$$P(\mathrm{TT}) = 0.44208$$

Therefore,

$$P(R \mid \mathrm{TT}) = \frac{\frac{81}{100} \times \frac{1}{4}}{0.44208} \approx 0.45806$$

Question 2 and 3 (Amazon Review Classification)

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import os
print(os.getcwd()) # to learn the location in order to store my dataset in the␣
↪same directory...
```

```
C:\Users\kadir\Desktop\4 1\CS 464
```

[2]:
```
start_time = datetime.datetime.now()
file = open("x_train.csv", "r")
dictionary_for_x = file.readline().split(" ")
```

[3]:
```
x_train = pd.read_csv("x_train.csv", delimiter = ",")
y_train = pd.read_csv("y_train.csv", delimiter = ",", header = None)
x_test = pd.read_csv("x_test.csv", delimiter = ",")
y_test = pd.read_csv("y_test.csv", delimiter = ",", header = None)
```

QUESTION 3.1

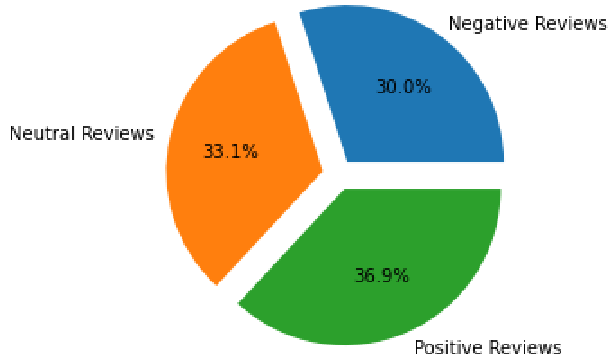
1. What are the percentages of each category in the y_train.csv y_test.csv? Draw a pie chart showing percentages.

[4]:
```
y_train_frequency = y_train[0].value_counts().sort_index()
y_test_frequency = y_test[0].value_counts().sort_index()
print(y_train_frequency)
print(y_test_frequency)
```

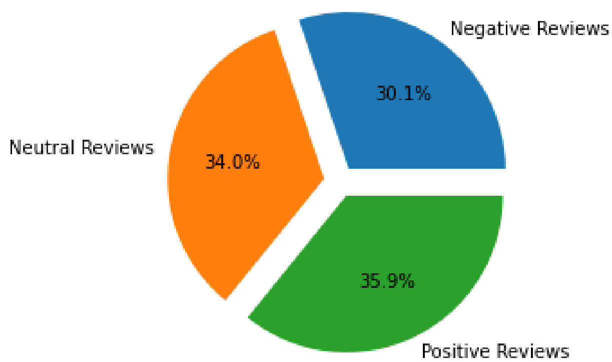
```
0    689
1    762
2    849
Name: 0, dtype: int64
0    211
1    238
2    251
Name: 0, dtype: int64
```

[5]:
```
def plot_pie(frequencies):
    class_labels_given = ["Negative Reviews", "Neutral Reviews", "Positive␣
↪Reviews"]
    explodes = 3 * [0.1]
    plt.pie(frequencies, labels = class_labels_given, explode = explodes,␣
↪autopct = '%1.1f%%')
    plt.show
```

[6]:
```
plot_pie(y_train_frequency)
```



[7]:
```
plot_pie(y_test_frequency)
```



2. What is the prior probability of each class?

[8]:
```
priors = (y_train_frequency / y_train_frequency.sum())
priors # Sum of priors becomes 1, as expected...
```

[8]:
```
0    0.299565
1    0.331304
2    0.369130
Name: 0, dtype: float64
```

[9]:
```
percentages_train = priors.to_numpy() # to use in Multinomial and Bernoulli␣
↪Document Models...
```

3. Is the training set balanced or skewed towards one of the classes? Do you think having an imbalanced training set affects your model? If yes, please explain how it can affect the model briefly. *### Answer:* **The training set is balanced since the priors are close to each other, so it is not skewed towards one of the classes. Having an imbalanced training set leads to the model predicting the majority class. For example, the prior probability of the class of 2, equivalent to Positive Reviews, would become $\frac{1}{2}$60 (just an assumption), and then we would say that our model is imbalanced.**

---

4. How many times do the words "good" and "bad" appear in the training documents with the label "positive", including multiple occurrences, and what is the log ratio of their occurrences within those documents, i.e, ln(P(good | Y = positive)) and ln(P(bad | Y = positive))?

[10]:
```
sum_of_good = x_train[y_train[0] == 2]["good"].sum()
sum_of_bad = x_train[y_train[0] == 2]["bad"].sum()
total_words_in_positive_as_2 = x_train[y_train[0] == 2].sum().sum()

good_given_positive_as_2 = sum_of_good / total_words_in_positive_as_2
bad_given_positive_as_2 = sum_of_bad / total_words_in_positive_as_2

log_version_good = np.log(good_given_positive_as_2)
log_version_bad = np.log(bad_given_positive_as_2)

print(f"Number of occurrences of 'good': {sum_of_good}")
print(f"Number of occurrences of 'bad': {sum_of_bad}")
print(f"ln(P(good | Y = positive)) = {log_version_good}")
print(f"ln(P(bad | Y = positive)) = {log_version_bad}")
```

```
Number of occurrences of 'good': 207
Number of occurrences of 'bad': 12
ln(P(good | Y = positive)) = -4.287609775546563
ln(P(bad | Y = positive)) = -7.135421919023932
```

QUESTION 3.2 and QUESTION 3.3

Training Multinomial Document Model with results given as without and with smoothing (alpha = 0, alpha = 1, respectively).

[11]:
```
def train_multinomial(x_train, y_train, alpha=0):
    x_train_np = x_train.to_numpy()
    y_train_np = y_train.to_numpy()
    no_of_class = len(np.unique(y_train))
    no_of_features = x_train.shape[1]
    likelihoods = np.zeros((no_of_class, no_of_features))


    for document in range(no_of_class):
        word_counts = np.zeros(no_of_features) + alpha
        count_document = alpha * no_of_features


        for i in range(len(y_train_np)):
            if y_train_np[i] == document:
                sample_features = np.array(x_train_np[i], dtype=float).flatten()
                word_counts += sample_features
                count_document += sample_features.sum()

        likelihoods[document] = np.log(word_counts) - np.log(count_document)
        likelihoods[document][likelihoods[document] == -np.inf] = -1e12

    return likelihoods

def posterior_multinomial(x_test, log_prior, log_likelihoods):
    predictions = []
    no_of_classes = log_prior.shape[0]
    for _, document in x_test.iterrows():
        max_log_prob = -np.inf
        max_index = -1
        for c in range(no_of_classes):
            log_prob = log_prior[c] + np.dot(document, log_likelihoods.iloc[c, :
↪])
            if log_prob > max_log_prob:
                max_log_prob = log_prob
                max_index = c
        predictions.append(max_index)
    return np.array(predictions)

def accuracy(test, predicted):
    correct = 0
    for i in range(len(test)):
        if (test[i] == predicted[i]):
```

```
            correct += 1

    return (correct / len(test))
```

[12]:
```
likelihoods = train_multinomial(x_train, y_train)
predictions = posterior_multinomial(x_test, np.log(percentages_train), pd.
 ↪DataFrame(likelihoods))
output = accuracy(y_test.to_numpy(), predictions)
print(f"Accuracy for the Multinomial Model without Smoothing: {output:.3f}")
print(f"%{(0.576 / output) * 100:.1f}") # To show how my output is close to the␣
 ↪expected in the homework document...
```

```
C:\Users\kadir\AppData\Local\Temp\ipykernel_21544\3528263612.py:20:
RuntimeWarning: divide by zero encountered in log
  likelihoods[document] = np.log(word_counts) - np.log(count_document)
```

```
Accuracy for the Multinomial Model without Smoothing: 0.581
%99.1
```

[13]:
```
likelihoods = train_multinomial(x_train, y_train, 1)
predictions = posterior_multinomial(x_test, np.log(percentages_train), pd.
 ↪DataFrame(likelihoods))
output = accuracy(y_test.to_numpy(), predictions)
print(f"Accuracy for the Multinomial Model with smoothing (alpha = 1): {output:.
 ↪3f}")
print(f"%{(0.633 / output) * 100:.1f}") # To show how my output is close to the␣
 ↪expected in the homework document...
```

```
Accuracy for the Multinomial Model with smoothing (alpha = 1): 0.649
%97.6
```

QUESTION 3.4

Training Bernoulli Document Model…

[14]:
```
def train_bernoulli(x_train, y_train, alpha=1):
    x_train_np = x_train.to_numpy()
    y_train_np = y_train.to_numpy()

    no_of_class = len(np.unique(y_train_np))
    no_of_features = x_train.shape[1]

    likelihoods = np.zeros((no_of_class, no_of_features))

    for document in range(no_of_class):
        word_counts = np.ones(no_of_features)
        count_document = (y_train[0] == document).sum()

        for i in range(len(y_train_np)):
```

```
            if y_train_np[i] == document:
                word_counts += (x_train_np[i] != 0).astype(int)

        likelihoods[document] = np.log(word_counts / (count_document + 2))
        likelihoods[document][likelihoods[document] == -np.inf] = -1e12

    return likelihoods

def posterior_bernoulli(x_test, log_prior, log_likelihood):
    predictions = []
    for _, row in x_test.iterrows():
        values = []
        for document_type in range(log_prior.shape[0]):
            log_prob = log_prior[document_type]
            log_prob += np.sum(log_likelihood[document_type][row == 1])
            log_prob += np.sum(np.log(1 - np.
 ↪exp(log_likelihood[document_type][row == 0])))
            values.append(log_prob)
        predictions.append(np.argmax(values))
    return np.array(predictions)
```

[15]:
```
x_train[x_train > 0] = 1
x_test[x_test > 0] = 1

likelihoods = train_bernoulli(x_train, y_train)
predictions = posterior_bernoulli(x_test, np.log(percentages_train), likelihoods)
output = accuracy(y_test.to_numpy(), predictions)
print(f"Accuracy for the Bernoulli Model with smoothing (alpha = 1):  {output:.
 ↪3f}")
print(f"%{(output / 0.643) * 100:.1f}") # To show how my output is close to the␣
 ↪expected in the homework document...
```

```
Accuracy for the Bernoulli Model with smoothing (alpha = 1):  0.641
%99.8
```

[16]:
```
# Compute the run-time
end_time = datetime.datetime.now()
elapsed_time = end_time - start_time
print(f"Run-Time for three questions : {elapsed_time.seconds} seconds")
```

```
Run-Time for three questions : 22 seconds
```

[ ]:
```
```