BILKENT UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
CS464 HOMEWORK 1

BERAT BİÇER
21503050

Nov. 4, 2018
ANKARA; TURKEY

QUESTION 1

Question 1.1

$P(Machine\ 2 \mid Loss) = P(Machine\ 2,\ Loss) \div P(Loss)$

$P(Loss) = P(Machine\ 1,\ Loss) + P(Machine\ 2,\ Loss)$

$P(Machine\ 2 \mid Loss) = (900 / 1130) \div ((135 / 200) + (900 / 1130))$

≈ 0.54

Note that these values belong to “You” in the table.

Question 1.2

You:

$$P(Loss \mid Machine\ 1) = 60 / (40 + 60) = 0.6$$
$$P(Loss \mid Machine\ 2) = 828 / (828 + 212) \approx 0.8$$

Friend:

$$P(Loss \mid Machine\ 1) = 75 / (25 + 75) = 0.25$$
$$P(Loss \mid Machine\ 2) = 72 / (18 + 72) = 0.8$$

So, probability of “You” losing at machine 1 is less than probability of “Friend” losing at machine 1, whereas probabilities of “You” and “Friend” losing at machine 2 are approximately equal.

Question 1.3

Assuming we merge the machines by adding #wins and losses with respect to each player,

$$P(Loss \mid You) = 888 / (252 + 888) \approx 0.22$$
$$P(Loss \mid Friend) = 147 / (33 + 147) \approx 0.18$$

This implies “You” is more likely to win compared to “Friend”.

Question 1.4

Sequence $Loss,\ Win,\ Win,\ Loss$ can only be generated as follows:

$$P(Loss \mid You),\ P(Win \mid Friend),\ P(Win \mid You),\ P(Loss \mid Friend).$$

Since these events are independent, probability of generating this sequence, named $P(s)$, is equal to

$$P(s) = P(Loss \mid You) \times P(Win \mid Friend) \times P(Win \mid You) \times P(Loss \mid Friend)$$
$$= 0.6 \times 0.25 \times 0.4 \times 0.75$$
$$= 0.045$$

QUESTION 2

Question 2.1

Since $P(b = 5)$ and $P(r = 5)$ are independent events, we can write

$$P(b = 5,\ r = 5 \mid C) = \frac{P(b=5, C) \times P(r=5, C)}{P(C)}$$
$$= \frac{1/4 \times 1/4}{4/6 \times 4/6} = \frac{1}{16}$$

Question 2.2

Similar to Question 2.1, events are independent and we can write

$$P(b = 5,\ r = 5 \mid D) = \frac{P(b=5, D) \times P(r=5, D)}{P(D)}$$
$$= \frac{1/3 \times 1/3}{1/2 \times 1/2} = \frac{4}{9}$$

Question 2.3

Difference between Question 2.1 and Question 2.2 is related to the prior information, C and D respectively. Prior assumption generates a conditional probability, and thus even if expected outcome is common, probability values are different.

QUESTION 3

Question 3.1

Let $\lambda *$ be MLE estimator of λ where $X \sim Poisson(\lambda)$ and its probability distribution function $P(X = x) = (\lambda^x \times e^{-\lambda}) \div x!$. Then, log likelihood function is

$$\sum_i (x_i \times ln\lambda - \lambda - ln(x_i!)) = ln\lambda \times \sum_i (x_i) - n \times \lambda - \sum_i ln(x_i!).$$

Let this be A. To maximize value of A, we need to take its partial derivative with respect to λ and set it to 0:

$$f'_\lambda(A) = \frac{1}{\lambda} \times \sum_i (x_i) - n = 0 \Rightarrow \lambda * = \sum_i (x_i) \div n = \overline{X}.$$

Question 3.2

To calculate MAP of λ, denoted as λ', we can say

$$argmax \frac{\prod_i P(X_i \mid \lambda) \times g(\lambda)}{P(X_1, .., X_n)} = argmax \prod_i P(X_i \mid \lambda) \times g(\lambda)$$

Since $P(X_1, \ldots, X_n)$ is constant.Let the value above be A. Then,

$$ln\, A = ln\lambda \times \sum_i (x_i) - n \times \lambda - \sum_i ln(x_i!) + ln\, k - (k + 1) \times ln\, \lambda$$

To maximize it, we set its partial derivative with respect to λ to 0:

$$f'_\lambda(A) = \frac{1}{\lambda} \times \sum_i (x_i) - n - \frac{k+1}{\lambda} = 0 \Rightarrow \lambda' = \overline{X} - \frac{k+1}{n}$$

Lastly, $\lambda > 1 \Rightarrow \overline{X} - \frac{k+1}{n} \geq 1$. Assuming $\overline{X} = 0$ and since $n > 0$, we get

$$- \frac{k+1}{n} - 1 \geq 0 \Rightarrow -k - 1 - n \geq 0$$

$$k \leq -n - 1$$

Question 3.3

MLE estimator of λ, denoted as $\lambda*$ from Question 3.1, is written as $\lambda* = \overline{X}$. To calculate MAP estimator of λ, denoted as λ' where prior distribution of λ is given by $\lambda \sim Pareto(x \mid k, 1)$, we follow similar steps to those in Question 3.2. Note that $g(\lambda) = Uniform(a, b)$. Then, log likelihood function becomes

$$ln\lambda \times \sum_i (x_i) - n \times \lambda - \sum_i ln(x_i!) + ln\, U(a, b)$$

To find MAP estimator of λ, we set the partial derivative of the quantity above with respect to λ to 0:

$$\frac{1}{\lambda} \times \sum_i (x_i) - n + f_\lambda'(U(a, b)) = 0 \Rightarrow \lambda' = \overline{X} + \frac{\lambda}{n} \times f_\lambda'(U(a, b))$$

From equation of λ', we can see that $\lambda* = \lambda' \Leftrightarrow f_\lambda'(U(a, b)) = 0$. Since $U(a, b) = 0\, for\ \forall a, b\ where\ b > a$, we are done.

QUESTION 4

Question 4.1

When maximizing a quantity that is in fractal form, maximizing the quantity itself is equivalent to maximizing its numerator as follows:

$$argmax\ \frac{A}{B} = argmax\ A$$

where A, B are some quantities and $B \neq 0$. Also, in the program, we do not require exact probability values. A relationship showing which label is more probable is sufficient to make predictions. Therefore, we can remove the divisor.

Question 4.2

We need to estimate, for each word in the vocabulary, the following:

- $\theta j \mid y = 0$, probability that a particular word in a medical email will be in the j-th word of the vocabulary, $P(Xj \mid Y = 0)$
- $\theta j \mid y = 1$, probability that a particular word in a space email will be in the j-th word of the vocabulary, $P(Xj \mid Y = 1)$

And lastly, we need to estimate $\pi\,(y = 0)$, probability that any particular email will be a space email. Knowing this value, $\pi\,(y = 1)$ can be calculated without estimation. Thus, total #parameters to estimate is $2 * V + 1$ where V is vocabulary size, in this case 26507.

Question 4.3

There are 800 medical and 800 space emails in the training data, or %50 - %50 distribution. This implies training data has a balanced class distribution.

Question 4.4

Final accuracy is **0.1675.** The classifier predicted poorly for most cases. Total number of false predictions is **333**. MLE is a poor choice of estimate since for every word classifier has not seen before log likelihood value is negative infinity (log 0), and each time a negative infinity appears prediction is medical. That's why MLE is a poor estimator for this case.

```
import numpy as np

train_features = np.loadtxt("dataset/question-4-train-features.csv",
dtype='i', delimiter=',')
train_labels = np.loadtxt("dataset/question-4-train-labels.csv", dtype='i',
delimiter=',')
test_features = np.loadtxt("dataset/question-4-test-features.csv", dtype='i',
delimiter=',')
test_labels = np.loadtxt("dataset/question-4-test-labels.csv", dtype='i',
delimiter=',')

vocabulary_size = len(train_features[0])
N = len(train_features)
sum_T_j_y_zero = 0
sum_T_j_y_one = 0
N_one = 0
N_zero = 0

# 1 space, 0 medical

for i in range(N):
    if train_labels[i] == 0: # medical
        N_zero = N_zero + 1
        sum_T_j_y_zero = sum_T_j_y_zero + train_features[i].sum(axis = 0)
    else: # space
        N_one = N_one + 1
        sum_T_j_y_one = sum_T_j_y_one + train_features[i].sum(axis = 0)

T_j_y_zero = np.zeros(vocabulary_size)
T_j_y_one = np.zeros(vocabulary_size)

for i in range(vocabulary_size):
    for j in range(N):
        if train_labels[j] == 0:
```

```
            T_j_y_zero[i] = T_j_y_zero[i] + train_features[j][i]
        else:
            T_j_y_one[i] = T_j_y_one[i] + train_features[j][i]

theta_j_y_zero = np.zeros(vocabulary_size)
theta_j_y_one = np.zeros(vocabulary_size)
for i in range(vocabulary_size):
    theta_j_y_zero[i] = float(T_j_y_zero[i] / sum_T_j_y_zero)
    theta_j_y_one[i] = float(T_j_y_one[i] / sum_T_j_y_one)

    if theta_j_y_zero[i] != 0:
        theta_j_y_zero[i] = np.log(theta_j_y_zero[i])

    if theta_j_y_one[i] != 0:
        theta_j_y_one[i] = np.log(theta_j_y_one[i])

correct_prediction_count = 0
for i in range(len(test_features)):
    weighted_theta_j_y_zero = float(0)
    weighted_theta_j_y_one = float(0)
    for j in range(vocabulary_size):
        weighted_theta_j_y_zero = weighted_theta_j_y_zero +
float(theta_j_y_zero[j] * test_features[i][j])
        weighted_theta_j_y_one = weighted_theta_j_y_one +
float(theta_j_y_one[j] * test_features[i][j])

    prediction_zero = np.log(float(N_zero / N)) + weighted_theta_j_y_zero
    prediction_one = np.log(float(N_one / N)) + weighted_theta_j_y_one
    prediction = 0 if prediction_zero >= prediction_one else 1
    correct_prediction_count = correct_prediction_count + 1 if prediction ==
test_labels[i] else correct_prediction_count

accuracy = float(correct_prediction_count / len(test_features))
print("Accuracy -> " + str(accuracy))
print("False predictions -> " + str(len(test_features) -
correct_prediction_count))
```

Question 4.5

Final accuracy is **0.9675**, whereas number of false predictions is **13**. Code for this part is as follows:

```
import numpy as np

train_features = np.loadtxt("dataset/question-4-train-features.csv",
dtype='i', delimiter=',')
```

```
train_labels = np.loadtxt("dataset/question-4-train-labels.csv", dtype='i',
delimiter=',')
test_features = np.loadtxt("dataset/question-4-test-features.csv", dtype='i',
delimiter=',')
test_labels = np.loadtxt("dataset/question-4-test-labels.csv", dtype='i',
delimiter=',')

vocabulary_size = len(train_features[0])
N = len(train_features)
sum_T_j_y_zero = 0
sum_T_j_y_one = 0
N_one = 0
N_zero = 0

# 1 space, 0 medical

for i in range(N):
    if train_labels[i] == 0: # medical
        N_zero = N_zero + 1
        sum_T_j_y_zero = sum_T_j_y_zero + train_features[i].sum(axis = 0)
    else: # space
        N_one = N_one + 1
        sum_T_j_y_one = sum_T_j_y_one + train_features[i].sum(axis = 0)

T_j_y_zero = np.zeros(vocabulary_size)
T_j_y_one = np.zeros(vocabulary_size)

sum_T_j_y_zero = sum_T_j_y_zero + vocabulary_size
sum_T_j_y_one = sum_T_j_y_one + vocabulary_size

for i in range(vocabulary_size):
    for j in range(N):
        if train_labels[j] == 0:
            T_j_y_zero[i] = T_j_y_zero[i] + train_features[j][i]
        else:
            T_j_y_one[i] = T_j_y_one[i] + train_features[j][i]

theta_j_y_zero = np.zeros(vocabulary_size)
theta_j_y_one = np.zeros(vocabulary_size)
for i in range(vocabulary_size):
    theta_j_y_zero[i] = float((T_j_y_zero[i] + 1) / sum_T_j_y_zero)
    theta_j_y_one[i] = float((T_j_y_one[i] + 1) / sum_T_j_y_one)

    if theta_j_y_zero[i] != 0:
        theta_j_y_zero[i] = np.log(theta_j_y_zero[i])

    if theta_j_y_one[i] != 0:
        theta_j_y_one[i] = np.log(theta_j_y_one[i])
```

```
correct_prediction_count = 0
for i in range(len(test_features)):
    weighted_theta_j_y_zero = float(0)
    weighted_theta_j_y_one = float(0)
    for j in range(vocabulary_size):
        weighted_theta_j_y_zero = weighted_theta_j_y_zero +
float(theta_j_y_zero[j] * test_features[i][j])
        weighted_theta_j_y_one = weighted_theta_j_y_one +
float(theta_j_y_one[j] * test_features[i][j])

    prediction_zero = np.log(float(N_zero / N)) + weighted_theta_j_y_zero
    prediction_one = np.log(float(N_one / N)) + weighted_theta_j_y_one
    prediction = 0 if prediction_zero >= prediction_one else 1
    correct_prediction_count = correct_prediction_count + 1 if prediction ==
test_labels[i] else correct_prediction_count

accuracy = float(correct_prediction_count / len(test_features))
print("Accuracy -> " + str(accuracy))
print("False predictions -> " + str(len(test_features) -
correct_prediction_count))
```

Question 4.6

In this section, Mutual information scores are added to the code in Question 4.5.

```
import numpy as np

train_features = np.loadtxt("dataset/question-4-train-features.csv",
dtype='i', delimiter=',')
train_labels = np.loadtxt("dataset/question-4-train-labels.csv", dtype='i',
delimiter=',')
test_features = np.loadtxt("dataset/question-4-test-features.csv", dtype='i',
delimiter=',')
test_labels = np.loadtxt("dataset/question-4-test-labels.csv", dtype='i',
delimiter=',')

vocabulary_size = len(train_features[0])
N = len(train_features)
sum_T_j_y_zero = 0
sum_T_j_y_one = 0
N_one = 0
N_zero = 0

# 1 space, 0 medical

for i in range(N):
    if train_labels[i] == 0: # medical
```

```
        N_zero = N_zero + 1
        sum_T_j_y_zero = sum_T_j_y_zero + train_features[i].sum(axis = 0)
    else: # space
        N_one = N_one + 1
        sum_T_j_y_one = sum_T_j_y_one + train_features[i].sum(axis = 0)

T_j_y_zero = np.zeros(vocabulary_size)
T_j_y_one = np.zeros(vocabulary_size)

sum_T_j_y_zero = sum_T_j_y_zero + vocabulary_size
sum_T_j_y_one = sum_T_j_y_one + vocabulary_size

for i in range(vocabulary_size):
    for j in range(N):
        if train_labels[j] == 0:
            T_j_y_zero[i] = T_j_y_zero[i] + train_features[j][i]
        else:
            T_j_y_one[i] = T_j_y_one[i] + train_features[j][i]

theta_j_y_zero = np.zeros(vocabulary_size)
theta_j_y_one = np.zeros(vocabulary_size)
for i in range(vocabulary_size):
    theta_j_y_zero[i] = float((T_j_y_zero[i] + 1) / sum_T_j_y_zero)
    theta_j_y_one[i] = float((T_j_y_one[i] + 1) / sum_T_j_y_one)

    if theta_j_y_zero[i] != 0:
        theta_j_y_zero[i] = np.log(theta_j_y_zero[i])

    if theta_j_y_one[i] != 0:
        theta_j_y_one[i] = np.log(theta_j_y_one[i])

correct_prediction_count = 0
for i in range(len(test_features)):
    weighted_theta_j_y_zero = float(0)
    weighted_theta_j_y_one = float(0)
    for j in range(vocabulary_size):
        weighted_theta_j_y_zero = weighted_theta_j_y_zero +
float(theta_j_y_zero[j] * test_features[i][j])
        weighted_theta_j_y_one = weighted_theta_j_y_one +
float(theta_j_y_one[j] * test_features[i][j])

    prediction_zero = np.log(float(N_zero / N)) + weighted_theta_j_y_zero
    prediction_one = np.log(float(N_one / N)) + weighted_theta_j_y_one
    prediction = 0 if prediction_zero >= prediction_one else 1
    correct_prediction_count = correct_prediction_count + 1 if prediction ==
test_labels[i] else correct_prediction_count

accuracy = float(correct_prediction_count / len(test_features))
```

```
print("Accuracy -> " + str(accuracy))
print("False predictions -> " + str(len(test_features) -
correct_prediction_count))

# Mutual information between class variable and features
mi = {}
for i in range(vocabulary_size):
    N00 = float(0)
    N01 = float(0)
    N10 = float(0)
    N11 = float(0)
    for j in range(N):
        if train_features[j][i] == 0 and train_labels[j] == 0: # N00 or N01
            N00 = N00 + 1
        if train_features[j][i] == 0 and train_labels[j] != 0:
            N01 = N01 + 1
        if train_features[j][i] != 0 and train_labels[j] == 0:
            N10 = N10 + 1
        if train_features[j][i] != 0 and train_labels[j] != 0:
            N11 = N11 + 1

    first_term = float((N11 / N) * (np.log2(N * N11) - np.log2(N10 + N11) -
np.log2(N01 + N11)))
    second_term = float((N01 / N) * (np.log2(N * N01) - np.log2(N00 + N01) -
np.log2(N01 + N11)))
    third_term = float((N10 / N) * (np.log2(N * N10) - np.log2(N10 + N11) -
np.log2(N00 + N10)))
    forth_term = float((N00 / N) * (np.log2(N * N00) - np.log2(N00 + N01) -
np.log2(N00 + N10)))
    mi[i] = float(first_term + second_term + third_term + forth_term)

sorted_mi = sorted(mi.items(), key = lambda kv: kv[1], reverse = True)
print(sorted_mi)
```