POLYTECHNIQUE MONTRÉAL

LOG3430 - MÉTHODES DE TEST ET DE VALIDATION DU LOGICIEL

LABORATOIRE 3

TESTS D'INTERACTIONS

Département de génie informatique et génie logiciel

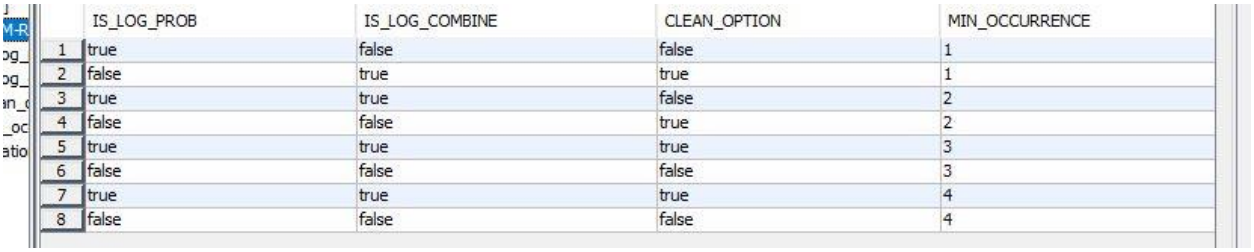
Amir Cherkaoui Eddeqaqi - 1980843
Mike Folepe - 1894509
Etienne Chevalier Titteley – 2021242

Gr.02

Hiver 2022

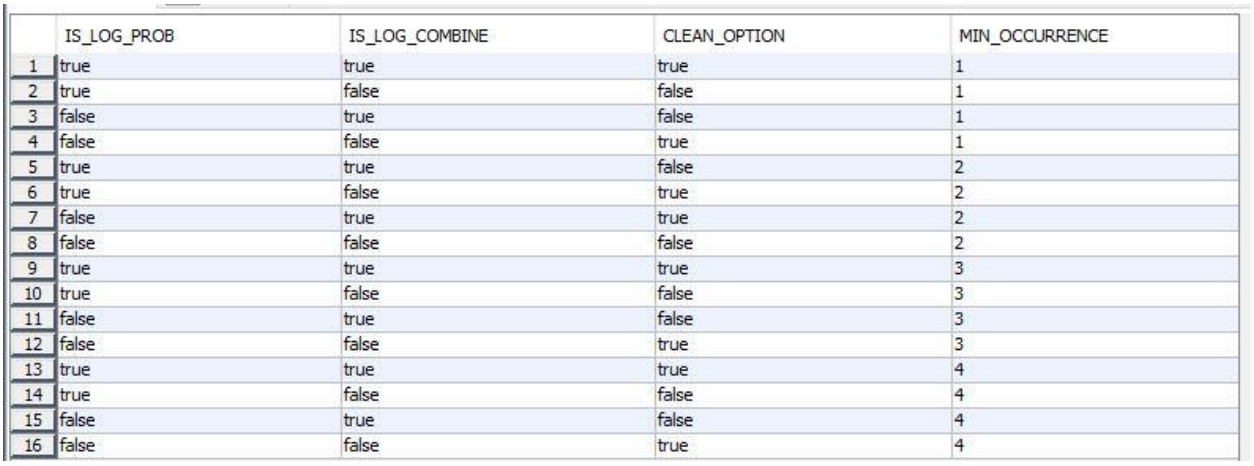
I- Taille des matrices MCA

- MCA (N ,t ,k ,v) = MCA (8, 2, 4, {2,2,2,4}) interactions doubles



	IS_LOG_PROB	IS_LOG_COMBINE	CLEAN_OPTION	MIN_OCCURRENCE
1	true	false	false	1
2	false	true	true	1
3	true	true	false	2
4	false	false	true	2
5	true	true	true	3
6	false	false	false	3
7	true	true	true	4
8	false	false	false	4

- MCA (N , t, k ,v) = MCA (16, 3, 4, {2,2,2,4}) interactions triples



	IS_LOG_PROB	IS_LOG_COMBINE	CLEAN_OPTION	MIN_OCCURRENCE
1	true	true	true	1
2	true	false	false	1
3	false	true	false	1
4	false	false	true	1
5	true	true	false	2
6	true	false	true	2
7	false	true	true	2
8	false	false	false	2
9	true	true	true	3
10	true	false	false	3
11	false	true	false	3
12	false	false	true	3
13	true	true	true	4
14	true	false	false	4
15	false	true	false	4
16	false	false	true	4

- MCA (N, t, k ,v) = MCA (32, 4, 4, {2,2,2,4}) interactions quadruples

	IS_LOG_PROB	IS_LOG_COMBINE	CLEAN_OPTION	MIN_OCCURRENCE
1	true	true	true	1
2	true	true	false	1
3	true	false	true	1
4	true	false	false	1
5	false	true	true	1
6	false	true	false	1
7	false	false	true	1
8	false	false	false	1
9	true	true	true	2
10	true	true	false	2
11	true	false	true	2
12	true	false	false	2
13	false	true	true	2
14	false	true	false	2
15	false	false	true	2
16	false	false	false	2
17	true	true	true	3
18	true	true	false	3
19	true	false	true	3
20	true	false	false	3
21	false	true	true	3
22	false	true	false	3
23	false	false	true	3
24	false	false	false	3
25	true	true	true	4
26	true	true	false	4
27	true	false	true	4
28	true	false	false	4
29	false	true	true	4
30	false	true	false	4
31	false	false	true	4
32	false	false	false	4

II- Analyse des résultats

On va analyser le jeu de tests générés par notre matrice à trois interactions

Test	Is_log_prob	Is_log_combine	Clean_option	min_occurence	accuracy	Precision	Recall
#1	True	True	True(1)	1	0.89	0.94	0.75
#2	True	False	False(0)	1	0.89	0.94	0.75
#3	False	True	False(0)	1	0.89	0.94	0.75
#4	False	False	True(1)	1	0.89	0.94	0.75
#5	True	True	False(0)	2	0.81	0.7	0.86
#6	True	False	True(1)	2	0.81	0.7	0.86
#7	False	True	True(1)	2	0.81	0.7	0.86
#8	False	False	False(0)	2	0.81	0.7	0.86
#9	True	True	True(1)	3	0.69	0.57	0.86
#10	True	False	False(0)	3	0.69	0.57	0.86
#11	False	True	False(0)	3	0.69	0.57	0.86
#12	False	False	True(1)	3	0.69	0.57	0.86
#13	True	True	True(1)	4	0.58	0.47	0.86
#14	True	False	False(0)	4	0.58	0.47	0.86
#15	False	True	False(0)	4	0.58	0.47	0.86
#16	False	False	True(1)	4	0.58	0.47	0.86

a- Analyse complète

Selon les résultats des tests générés ci-dessus, on peut observer que lorsque les paramètres is_log_prob, is_log_combine et clean_option varient alors que le paramètre min_occurrence reste constant, les valeurs d'accuracy, de la précision ainsi que du recall sont inchangées. Ce qui implique que l'utilisation de la formule incluant la somme de "log" ne modifie pas la qualité des tests par rapport à la formule incluant des produits de probabilités pour le calcul ou la combinaison. Ainsi, seulement la valeur du paramètre min_occurrence influence la qualité des tests.

D'ailleurs, lorsque le nombre minimal requis d'occurrences pour considérer un mot augmente, l'accuracy et la précision du test diminue. Ceci peut être expliqué par le fait qu'il y aura moins de mots dans le dictionnaire lors de la création du vocabulaire, ainsi lorsqu'on compare les mots contenus dans l'email au vocabulaire, on risque d'ignorer plusieurs mots qui par exemple augmente la probabilité de spam, car on retient seulement les mots avec une certaine fréquence. Donc, quand le nombre d'occurrences minimal augmente, la marge d'erreur des calculs des probabilités de spam et ham augmente. Selon les équations suivantes :

Accuracy = true positive + true negative / (true positive + true negative + false positive + false negative)
Precision = true positive / (true positive + false positive)

On observe que l'augmentation du nombre minimal d'occurrences augmente le nombre de tests false positive et false negative.

En ce qui concerne la valeur du recall, on observe qu'elle est plus grande lorsque le nombre minimal d'occurrences est plus grand que 1. Ainsi selon l'équation suivante du recall :

Recall = true positive / (true positive + false negative)

On peut analyser que le nombre de tests "false negative" diminue pour un nombre minimal d'occurences plus grand que 1.

b- Meilleures configurations

Comme on peut dans le tableau, les meilleures configurations sont les tests #1, #2, #3 et #4. En effet, celles-ci ont une accuracy de 89%, une precision de 94%, et enfin un recall de 75%. Malgré le fait que le recall est plus haut dans d'autres tests, si l'on prend en compte la moyenne des 3 critères, les 4 premiers tests sont meilleurs. Cela peut s'expliquer par le fait que les 4 premiers tests évaluent une occurrence minimale de 1. La probabilité que le mot n'apparaisse qu'une fois et bien plus haute que la probabilité qu'il apparaisse 2, 3 ou 4 fois et c'est exactement ce que le test montre.

c- Pires configurations

Les tests #13,#14,#15,#16 sont les pires configurations car ils réduisent significativement la performance de notre système en observant les métriques dans le tableau. Ceci s'explique par le fait que le nombre minimal d'occurrence dans ces cas de tests est de 4. C'est-à-dire que le vocabulaire est créé en prenant seulement les probabilités pour les mots qui apparaissent un minimum de 4 fois. Ce qui fait que si on a d'autres mots qui pourraient faire en sorte que le message soit considéré plus un spam qu'un ham et vice-versa , ils ne seront pas pris en compte s'ils n'apparaissent pas au moins 4 fois, ce qui réduit donc significativement l'exactitude car le système ne peut pas clairement déterminer si un message est spam ou ham et précision car les probabilités qu'on obtient ne reflète pas en détails l'état des messages analysés par le système.

d- Défauts détectés dans le système

En exécutant le jeu de tests la première fois, ce dernier nous a relevé un défaut dans notre système qui était le fait que notre code ne prenait pas en compte toutes les combinaisons d'entrées pour les paramètres.

```
Traceback (most recent call last):
  File "main.py", line 83, in <module>
    exec_test_set()
  File "main.py", line 70, in exec_test_set
    renege.classify_emails(bool(test_case[0]), bool(test_case[1]), bool(test_case[2]))
  File "M:\LOG3430\TP3\renege.py", line 28, in classify_emails
    raise e
  File "M:\LOG3430\TP3\renege.py", line 24, in classify_emails
    self.process_email(self.get_email(),is_log_prob,is_log_combine,clean_option)
  File "M:\LOG3430\TP3\renege.py", line 53, in process_email
    is_spam = self.e_mail.is_spam(subject,body,is_log_prob,is_log_combine,clean_option)
  File "M:\LOG3430\TP3\email_analyzer.py", line 41, in is_spam
    p_sub_spam_text = 0 if p_subject_spam == 0 else k * math.log10(p_subject_spam)
ValueError: math domain error
PS M:\LOG3430\TP3>
```

On voit bien avec le message d'erreur que notre système ne prenait pas en compte certains cas limites comme la combinaison de is_log_prob = true et is_log_combine = true et faisait donc deux fois le logarithme, ce qui a généré une erreur.

Voici une capture d'écran du code de la fonction is_spam qui générait ce message d'erreur

```python
def is_spam(self, subject_orig, body_orig, is_log_prob, is_log_combine):
    '''
    Description: fonction pour verifier si e-mail est spam ou ham,
    en calculant les probabilites d'etre spam et ham,
    en fonction du sujet et du texte d'email.
    Sortie: 'True' - si l'email est spam, 'False' - si email est ham.
    '''
    # Clean email's subject and body
    email_subject = self.clean_text(subject_orig)
    email_body    = self.clean_text(body_orig)

    # Get the spam/ham probabilities

    # On pose k = 0.6 comme donnée dans l'énoncé pour le coefficient de p_subject.
    k = 0.6
    if is_log_prob:
        p_subject_spam, p_subject_ham = self.spam_ham_log_subject_prob(email_subject)
        p_body_spam,    p_body_ham    = self.spam_ham_log_body_prob(email_body)
    else:
        p_subject_spam, p_subject_ham = self.spam_ham_subject_prob(email_subject)
        p_body_spam,    p_body_ham    = self.spam_ham_body_prob(email_body)


    if is_log_combine:
        p_sub_spam_text = 0 if p_subject_spam == 0 else k * math.log10(p_subject_spam)
        p_body_spam_text = 0 if p_body_spam == 0 else (1 - k)* math.log10(p_body_spam)
        p_spam = math.pow(10, p_sub_spam_text + p_body_spam_text)

        p_sub_ham_text = 0 if p_subject_ham == 0 else k * math.log10(p_subject_ham)
        p_body_ham_text = 0 if p_body_ham == 0 else (1 - k)* math.log10(p_body_ham)
        p_ham = math.pow(10, p_sub_ham_text + p_body_ham_text)
    else:
        p_spam = k * p_subject_spam + (1 - k)* p_body_spam
        p_ham  = k * p_subject_ham  + (1 - k) * p_body_ham



    # Compute the merged probabilities


    # Decide is the email is spam or ham
    if p_spam > p_ham:
        return True
    else:
        return False
```

Le code suivant montre la solution à ce défaut rencontré

```
    if is_log_prob:
        # Si on choisit d'utiliser les logs,On calcule les logs des probabilités avec des fonctions que nous avons rajouté
        log_p_subject_spam, log_p_subject_ham = self.spam_ham_log_subject_prob(
            email_subject)
        log_p_body_spam,    log_p_body_ham = self.spam_ham_log_body_prob(
            email_body)
    else:
        #Si on choisit l'option contraire , On calcule les probabiltés sans les logs
        p_subject_spam, p_subject_ham = self.spam_ham_subject_prob(
            email_subject)
        p_body_spam,    p_body_ham = self.spam_ham_body_prob(email_body)

    # On pose k = 0.6 comme donnée dans l'énoncé pour le coefficient de p_subject.
    k = 0.6
    if is_log_combine:
        # L'option de combinaison des logs est choisi

        if is_log_prob:
            # On utilise les logs des probabilités déjà calculé dans la formule de la combinaison donnée dans le TP
            # logP(spam|text) = 0.6 * logP(spam sub|text) + 0.4 * logP(spam body|text)
            p_sub_spam_text = k * log_p_subject_spam
            p_body_spam_text = (1 - k) * log_p_body_spam

            # On retire le log du résultat pour l'utiliser dans la compairaison entre p_spam et p_ham
            p_spam = math.pow(10, p_sub_spam_text + p_body_spam_text)

            # Le même cheminement est utilisé pour les ham
            p_sub_ham_text = k * log_p_subject_ham
            p_body_ham_text = (1 - k) * log_p_body_ham

            p_ham = math.pow(10, p_sub_ham_text + p_body_ham_text)

        else:

            # On pose le résultat égale à 0 si la probabilité est égale à 0 pour éviter de faire le log de 0
            # On calcul ensuite la la combinaison avec la formule

            p_sub_spam_text = 0 if p_subject_spam == 0 else k * math.log10(p_subject_spam)
            p_body_spam_text = 0 if p_body_spam == 0 else (
                1 - k) * math.log10(p_body_spam)
            p_spam = math.pow(10, p_sub_spam_text + p_body_spam_text)

            p_sub_ham_text = 0 if p_subject_ham == 0 else k * math.log10(p_subject_ham)
            p_body_ham_text = 0 if p_body_ham == 0 else (
                1 - k) * math.log10(p_body_ham)
            p_ham = math.pow(10, p_sub_ham_text + p_body_ham_text)
    else:
        # L'option de combinaison des logs n'est pas choisi
        if is_log_prob:
            #Si on a calculé les logs des probabilités, on retire les logs pour faire le calcul
            p_subject_spam = math.pow(10, log_p_subject_spam)
            p_subject_ham = math.pow(10, log_p_subject_ham)
            p_body_spam = math.pow(10, log_p_body_spam)
            p_body_ham = math.pow(10, log_p_body_ham)

        # Le calcul est fait suivant la formule de l'énoncé du TP
        p_spam = k * p_subject_spam + (1 - k) * p_body_spam
        p_ham = k * p_subject_ham + (1 - k) * p_body_ham
```