IFT1169 – TRAVAIL PRATIQUE #4 – 30 mars 2010

« Recyclage ! »

Mohamed Lokbani

Équipes : le travail peut-être fait en binôme mais vous ne remettez qu'un travail par équipe.
Remise : une seule remise est à effectuer par voie électronique **le jeudi 22 avril 2010, 23h59 au plus tard, sans possibilités de prorogation**.
Conseils : n'attendez pas le dernier jour avant la remise pour engager votre travail. Vous n'aurez pas le temps nécessaire pour le réaliser.

But : ce TP a pour but de vous faire pratiquer la bibliothèque de modèles standard, les entrées, les sorties et les exceptions.

Énoncé : dans ce TP, vous devez écrire un programme C++ qui identifie l'auteur d'un texte.

En traitement automatique de la langue, on utilise souvent un modèle statistique pour identifier l'auteur ou la langue d'un texte. Ces modèles sont fondés sur l'observation de textes. Des textes qui se ressemblent doivent avoir forcément des caractéristiques similaires i.e : « ont le même auteur ou sont rédigés dans la même langue ».

Dans ce TP, nous allons utiliser la méthode de **bigrammes** pour identifier l'auteur d'une phrase[1]. Nous allons pour cela comparer la fréquence des occurrences de paires de mots consécutifs (bigrammes) dans une phrase avec la fréquence de ces bigrammes dans les textes de référence. Le travail se divise en 3 grandes étapes :

- **Étape 1 : construction des modèles de référence :** la première étape consiste à construire un modèle de langue pour chaque auteur. Ces modèles sont tout simplement des tables indiquant la fréquence de bigrammes (paires de caractères consécutifs). Pour construire ces modèles, les textes de référence vous sont fournis.

- **Étape 2 : évaluation du score de la phrase :** la seconde étape consiste à lire la phrase d'un auteur mystérieux et, bigramme par bigramme, calculer la probabilité ou le score que cette phrase soit de l'auteur a, b, ou c (etc.) en utilisant les modèles construits à l'étape précédente.

- **Étape 3 : comparaison des scores :** finalement, le score le plus élevé indiquera l'auteur du texte.

Description des étapes : voici une description de chaque étape.

Étape 1 : construction des modèles de référence: les modèles de langue sont tout simplement représentés par une table indiquant la probabilité des bigrammes.

- Un bigramme est formé de 2 mots consécutifs.
  Par exemple, si le texte de référence est tout simplement : ***une porte ouverte. une porte fermée!***
  Les bigrammes possibles sont :

  une,porte porte,ouverte ouverte,. .,une une,porte porte,fermée fermée,!

  Dans cet exemple, nous avons considéré les caractères de ponctuation comme des mots. De ce fait, le nombre de mots total dans la phrase est égal à 8.

[1] Ou un fichier texte. À voir pour cela la description de l'option « -f » en page 3 de ce document.

- La probabilité d'un bigramme est calculée ainsi:

$$\text{Prob(bigramme)} = \frac{\text{Freq(bigramme)} + \lambda}{N + \lambda B}$$

Où : Freq(bigramme) est le nombre de fois que ce bigramme apparaît dans le texte de référence
λ est une constante à fixer.
N est le nombre total de bigrammes dans le texte de référence.
B est une constante indiquant le nombre de bigrammes possibles (Nombre de mots+1)^2

Avec l'exemple précédent, on a:

```
N= 7                    /* 7 bigrammes dans la phrase */
λ = 0.5                 /* à titre d'exemple */
B = 81                  /* (8+1)^2      */
Freq(une,porte) = 2     /* 2 fois dans le texte */
Freq(ouverte,fermée) = 0  /* car le bigramme (ouverte,fermée) ne se retrouve pas dans le texte */
…
```

Le modèle à construire est donc une table avec les probabilités de tous les bigrammes :

```
Prob(une,porte) = (2 + 0.5) / (7 + 81 x 0.5) = 2.5 / 47.5 = 0.0526
Prob(ouverte,fermée) = (0 + 0.5) / (7 + 81 x 0.5) = 0.5 / 47.5 = 0.0105
```

etc.

Étape 2 : évaluation des scores de la phrase : votre programme doit ensuite lire une phrase tapée au clavier par l'utilisateur et calculer son score pour chaque auteur. Ces scores indiqueront la ressemblance de la phrase par rapport à chaque modèle. Pour calculer les scores (score_auteur1, score_auteur2, etc.), nous allons utiliser la méthode suivante :

Initialement :

```
score_auteur1 = 0
score_auteur2 = 0
```

Ensuite, pour chaque bigramme de la phrase, on calculera les scores comme suit :

```
score_auteur1 = score_auteur1 + log(modele_auteur1(bigramme))
score_auteur2 = score_auteur2 + log(modele_auteur2(bigramme))
```

Notez que la fonction `log` n'est autre que le logarithme.

Étape 3 : comparaison des résultats : finalement, nous allons tout simplement comparer la valeur de `score_auteur1` et `score_auteur2`. Le score le plus élevé indiquera l'auteur de la phrase. Si les scores sont identiques, vous indiquerez un message du genre "auteur non identifié".

Notez que, comme les virgules flottantes sont représentées de façon approximative, il ne faudra **jamais** faire la comparaison avec l'opérateur == entre 2 nombres représentés en virgule flottante. Il faut plutôt vérifier si la différence entre les 2 nombres est significative ou pas. Par exemple, pour comparer 2 doubles `d1` et `d2` l'on peut faire :

```
if (abs(d1 - d2) < 0.0001)
```

« 0.0001 » est une marge d'erreur que l'on se fixe (une petite constante, suffisante pour comparer les flottants). Si la valeur absolue de la différence entre d1 et d2 est inférieure à cette marge, alors l'on peut considérer que d1 est égal à d2.

Travail à réaliser :

1. La fonction « main » doit être développée dans le fichier « **tp4.cpp** »
2. Il peut être exécuté sous les formes suivantes :

a) option –a (construction des modèles) : pour construire les modèles, on utilisera l'option « **-a** », suivi de un ou plusieurs fichiers de référence.
tp4.exe –a texte1.txt, texte2.txt, etc.
Le résultat de ce calcul sera stocké dans (respectivement) **modele1.txt**, **modele2.txt** etc.
Et les noms des modèles seront stockés par défaut dans le fichier « **liste.txt** » comme suit :

modele1.txt	**où modele1.txt est le fichier de données correspondant au modèle 1**
modele2.txt	**où modele2.txt est le fichier de données correspondant au modèle 2**
….	**etc.**

On peut préciser un autre nom pour ce fichier avec l'option « **–n** » comme suit :

tp4.exe –n liste99.txt –a texte1.txt, texte2.txt, ...

b) déterminer l'auteur d'une phrase

tp4.exe ma_phrase_test

Pour pouvoir déterminer l'auteur de la phrase, vous avez besoin des fichiers de modèles. Les noms de ces derniers sont définis dans le fichier « **liste.txt** ». Lors de l'exécution de votre programme, ce dernier devra charger à partir du fichier « **liste.txt** » l'ensemble des modèles.

c) option –l (la liste des modèles) : elle permet de préciser un autre fichier de modèles. De ce fait, elle annule le cas b ci-dessus.

tp4.exe … –l liste00.txt …

Dans cet exemple, on prend en compte le fichier des modèles « **liste00.txt** » au lieu du fichier par défaut « **liste.txt** ». Cependant son format doit être le même que celui de « **liste.txt** ».

d) option –d (affichage simple ou détaillé) : votre programme devra permettre un affichage simple par défaut ou détaillé si l'option « **–d** » a été ajoutée dans la ligne de commande comme suit :

tp4.exe … –d …
Un affichage simple consiste à afficher l'auteur de la phrase entrée sur la ligne de commande. En cas d'égalité, comme précédemment mentionné, il faudra afficher "auteur non identifié".
Un affichage détaillé consiste à afficher la paire (auteur, score) dans un ordre décroissant par rapport au score calculé de cette dernière.

e) option –f (saisie d'un fichier) : jusqu'à présent, nous avons permis les tests réalisés sur des phrases passées sur la ligne de commande. Comme il n'est pas évident de déterminer l'auteur qui se cache derrière une très courte phrase test, il est important donc de tester sur un échantillon plus large. Cet échantillon est représenté par un fichier. De ce fait, votre programme devra permettre donc de les tester. Pour lui indiquer cela, il faudra utiliser l'option « –f » comme suit :

tp4.exe … –f test.txt

f) option –p (paramètre λ) : par défaut, la valeur de « **λ** » est égale à « 0.5 ». On peut préciser aussi une autre valeur à travers l'option « –p » comme suit :

tp4.exe –p 0.8 …

Fichiers fournis : ils représentent les fichiers de référence, à partir desquels vous allez calculer les modèles. Chaque fichier est associé à un auteur donné comme suit : (exemple du fichier « **fleurs.txt** »)

BAUDELAIRE ← auteur occupe toujours la première ligne du fichier
fleur
mal
poète
impeccable
très-vénéré
parfait
….

Le texte est donc découpé en mots. Chaque mot (y compris un mot composé) est situé sur une ligne. Ainsi le nombre total de mots dans un fichier nécessaire pour calculer B n'est autre que le nombre de lignes dans le fichier.

Indications : vu la quantité de données, vous allez être ramenés à utiliser une table de hachage, sinon vous allez avoir de sérieux problèmes au niveau de la gestion des ressources mémoires.
Pour cela, commencez par définir une classe paire, qui va contenir le bigramme (donc 2 strings). Vous devez lui définir les méthodes de calcul de hachage et de comparaison. Vous pouvez utiliser la formule suivante[2] :

Codehachage (la paire) = 3 * codehachage(1er élément de la paire) + 15 * codehachage(2e élément de la paire)

Hypothèses et contraintes :

- Pour commencer, vous devez déjà respecter les différentes contraintes de programmation précédemment décrites.
- C'est un travail à faire à **deux** ! Et deux ne signifie pas quatre ni dix. Plagiat équivaut à un 0 pour commencer.
- Se mettre à deux, signifie aussi relever le niveau de la réflexion dans le cadre de l'enrichissement de vos connaissances.
- Ce travail est un exercice algorithmique, donc il faut penser à l'optimiser!
- Il faudra vous assurer de respecter les noms de fichiers. Vous devez respecter aussi le format de l'affichage en sortie.

Remise : il est important de noter que votre TP sera compilé avec « gcc4.4.0 ». Si par choix vous décidez d'utiliser un autre compilateur, vérifiez que le code que vous avez produit (devant normalement fonctionner correctement chez vous) fonctionne aussi bien sur les ordinateurs de la DESI. Pour avoir la version du compilateur, utilisez la commande "**gcc –v**", qui devra donner le numéro de version « **4.4.0** ».

Par ailleurs, assurez-vous de la présence de l'option « pedantic » sur la ligne de compilation.

Pour éviter d'omettre un des fichiers associés à ce travail lors de la remise, nous vous conseillons de regrouper l'ensemble des fichiers dans un seul fichier compressé **tp4h10.zip** et de remettre ce dernier comme suit :

1. commencez d'abord par vous connecter sur la machine « **remise** » comme cela a été pratiqué dans la démo #01.
2. envoyez l'ensemble de vos fichiers par la procédure de remise électronique habituelle (Pour obtenir de l'aide sur cette commande, tapez dans un Xterm : man remise). Respectez les noms des fichiers.

   remise ift1169 1169tp4 tp4h10.zip

Attention: le nom du répertoire a changé par rapport aux tp#1 et tp#2. C'est « 1169tp4 ».

3. vérifiez que la remise s'est effectuée correctement.

   remise –v ift1169 1169tp4

[2] Moins efficace, mais elle fait l'affaire.

Barème : ce TP4 est noté sur 10 points.

Compilation et respect des spécifications	**1**
Codage, commentaires etc.	**3.5**
Rapport	**2.5**
Tests (fournis et non fournis)	**3**

En plus du précédent barème, vous risquez de perdre des points dans les cas suivants ….

- La non remise électronique (volontaire ou par erreur) est sanctionnée par la note 0.
- Les programmes ne contenant pas d'en-tête, -1 point.
- Un programme qui ne compile pas : 0.
- Un programme qui compile mais ne réalise pas les choses prévues dans la spécification : 0.
- Les avertissements (warnings) non corrigés : cela dépend de la quantité! À partir de -0.25 et plus.
- Le non respect du nom du fichier va générer une erreur de compilation donc un des points de la spécification n'a pas été respecté : 0.
- Aberration dans le codage : même si tous les chemins mènent à Rome, faites l'effort nécessaire pour éviter de prendre le plus long!

Automatisation de la correction : nous allons utiliser des scriptes pour compiler, exécuter et corriger votre programme. Assurez-vous de respecter les noms de fichiers.
Pour comparer votre sortie avec celle fournie en exemple, redirigez votre sortie dans un fichier avec le symbole « > » comme suit : « tp4.exe > masortie.txt ». Par la suite, comparez les deux sorties (obtenue et fournie) en vous aidant pour cela de la commande « diff » sous « MinGW/MSys » : « **diff –b –w –i –B sortietest.txt votresortie.txt** ». Sous « DOS », utilisez plutôt la commande « fc » tout en ajustant ses options. Si les fichiers sont identiques, vous n'obtiendrez rien en sortie. Sinon, vous obtiendrez les différences entre les deux fichiers. Pour plus de détails sur cette commande « unix », sur votre console « xterm » exécutez la commande suivante: « man diff ».
Si la différence est provoquée par des caractères non imprimables (« bizarroïdes »), il faudra vous assurer que le fichier texte est dans le bon format d'encodage. Pour cela, utilisez par exemple :

« Conversion dos à unix » : **dos2unix < entree > sortie**
« Conversion unix à dos » : **unix2dos < entree > sortie**

Par ailleurs, nous utiliserons d'autres jeux de test pour la correction.

Ainsi donc, au moment de la remise de votre travail, vous avez déjà une idée approximative de la note que vous allez obtenir. Si la commande « diff » ne signale pas de différences entre les deux fichiers, vous êtes sur la bonne voie pour obtenir une note élevée.

Des questions à propos de ce TP?

Une seule adresse : dift1169@iro.umontreal.ca

Pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1169] et une référence au **tp04**.

Mise à jour

30-03-2010 diffusion de l'énoncé.

Questions & Réponses :

- **À la page 2 de l'énoncé du TP, il est indiqué que B est une constante indiquant le nombre de bigrammes possibles (nombre de mots + 1) au carré. Plus loin, il est indiqué que B = (nombre de mots différents +1) au carré. Que doit-on utiliser ?**

Le nombre de bigrammes possibles = (nombre de mots (forcément) différents +1)^2.

- **Est-ce qu'il ne faudrait pas introduire la valeur absolue dans l'équation suivante:**
score_auteur1 = score_auteur1 + abs(log(modele_auteur1(bigramme)))?

Non. L'équation telle que donnée dans l'énoncé est correcte.

- **Quel est le format du fichier « liste.txt »?**

Il est disponible déjà à la page 3, paragraphe a).

- **Dans le cas où on va entrer la phrase « tp4.exe ma phrase teste » sur la console, est-ce que les espaces blancs doivent être remplacés par « _ »?**

Non. La syntaxe exacte est : « tp4.exe ma phrase teste ».

- **La fonction logarithmique utilisée pour calculer le score est-elle la même que la fonction log(...) définie dans « cmath »?**

Oui.

- **Est-ce que le score d'une phrase (ou d'un fichier test) est égal à la somme de logarithmes des probabilités pour tous les bigrammes qu'elle contienne? Ces probabilités se trouvent-elles dans les fichiers modèles?**

Oui, sauf dans le cas où un des bigrammes ne se trouve pas dans le modèle [voir exemple page 2: p(ouverte,fermée)]. Dans ce cas il faut calculer la probabilité du bigramme sachant que sa fréquence est nulle en tenant compte des paramètres du modèle sur lequel ce calcul va avoir lieu (constantes N et B).

- **Est ce que le bigramme « Ma,le » est le même que « ma,le » ou bien c'est un autre?**

Vous ne devrez pas avoir dans le texte un « Ma » et un « ma », sauf une erreur de programmation dans notre programme de lemmatisation!

- **Si dans le répertoire il existe déjà un fichier qui porte le nom « modele1.txt » est ce qu'il doit être « écrasé »?**

Plusieurs possibilités : traiter l'erreur avec une exception, ou bien écraser le fichier. Un peu de créativité ...

- **Le symbole « - » que l'on voit dans le fichier « arlequin.txt », par exemple, est-ce qu'il représente la fin d'un paragraphe?**

Il n'y a pas de notion de phrase ou de paragraphe. Il y a juste une suite de mots, un par ligne. À partir de cette suite, il faut construire les bigrammes.

- **Pour le paramètre « -a » est-ce que les noms de fichiers sont séparés avec une virgule et un espace ou bien des virgules uniquement ?**

Comme indiqué, les noms sont séparés par des virgules. Le "justifié" de « word » a mis son grain de sel dans la présentation de l'exemple!!!

- **Est-ce que l'ordre des paramètres est important? Par exemple, est-ce que « -n » doit venir nécessairement avant « -a » ?**

Dans ce cas, l'ordre ne compte pas. Vous pouvez traiter les options sur le volet, comme vous pouvez les traiter après avoir lu toute la ligne de commande. La seconde solution est "plus simple" à coder.

- **Doit-on valider la duplication de paramètres ? Par exemple: tp4.exe -a f1.txt, f2.txt -n f3.txt -a f4.txt**

Il n'y aura pas de duplication de paramètres dans les tests, même si en pratique il importe peu. En effet, on ne fait que lire les options ... donc la duplication ne rentre pas en jeu, sauf le cas suivant : « -n toto -n tata ... ». Il faut se décider à la fin prendre quoi « toto » ou « tata »?

- **Pourquoi suggérer les valeurs 3 et 15 comme clefs pour la table de Hashage ?**

Le choix vient de certaines expériences personnelles réalisées sur les bigrammes. Les valeurs 3 et 15 ne sont pas fixes. On a voulu juste distinguer entre les paires. On s'est dit pourquoi pas 3 et 15.

- **Doit-t-on tester si les paramètres peuvent être en lettres majuscules ?**

Non. Ils sont forcément en minuscules.

- **C'est quoi la différence entre « -n » et « -l » ?**

L'option « -n » permet de créer le fichier alors que l'option « -l » va le charger à partir de la valeur associée à cette option. Pratiquement l'option « -n » sert à écraser un fichier s'il existe, alors que l'option « -l » ne le fait pas.