Interaction entre gènes : extraction d'information d'un corpus de résumés d'articles de recherche

Franck Chantelot, Alexandre Claude, Vincent Zoonekynd

Mars 2004

Résumé

Les informations sur les intéractions entre gènes sont contenues dans des articles de recherche et pas encore facilement accessibles dans des bases de données : nous présentons et implémentons un moyen de transformer cette information textuelle non structurée en une base de données, facilement interrogeable.

1 Introduction

La fouille de texte (text mining) est un nouveau domaine qui vise à résoudre les problèmes de surcharge d'informations et à faciliter la recherche des connaissances cachées dans les documents. En effet, le volume d'informations disponibles croit de plus en plus vite notamment sur Internet où la plupart des informations sont sous forme textuelle (80%).

D'une part, les informations sont nombreuses et se présentent sous différentes formes (langages naturels et HTML) et d'autre part, elles ont différents supports (sites Web et bibliothèques numériques). Il devient de plus en plus difficile de trouver des informations correspondant au besoin d'un utilisateur ou d'en extraire des connaissances utiles. Ainsi la fouille de texte peut s'appliquer à beaucoup de domaines (l'analyse des e-mails, le suivi des profils utilisateurs, l'organisation des sites Web, etc.) dont la Biologie, en particulier pour la recherche d'information rapide dans des résumés de publications.

Ce projet se range dans le cadre du Text Mining : nous tentons d'extraire des informations sur l'interaction entre deux ou plusieurs gènes à partir de résumés bibliographiques PubMed. En effet, l'information biologique issue de l'expérience reste essentiellement diffusée à travers la publication des résultats dans les journaux scientifiques. De plus l'exploitation de l'information bibliographique est un aspect essentiel de l'activité du chercheur. Cette analyse demande un temps important pour l'extraction des données pertinentes expliquant ainsi cette volonté d'automatiser la tache.

Ce projet est articulé en deux axes principaux, l'analyse et l'extraction d'information (avec des aspects algorithmiques et statistiques) d'une part, le stockage et la restitution de l'information (base de données et Interface Homme-Machine (IHM)) d'autre part.

La première partie a consisté à transformer des données bibliographiques non structurées ou semi-structurées (textes en anglais) en données structurées utilisables pour une étude statistique (liste de mots). Cela a permis d'extraire une information pertinente.

La seconde partie du projet s'articule autour de la mise en place d'une base de données permettant de stocker les informations générées et de la création d'une interface graphique permettant d'interroger la base de données ainsi constituée.

Dans ce rapport, nous nous focalisons sur la première partie.

2 Données

Les données de la biologie se caractérisent par leur diversité et leur hétérogénéité. En effet le language naturel (utilisé dans les publications) est un support de communication dont il est difficile d'extraire de l'information de façon automatique. Il est donc nécessaire de modifier les données de départ avant de pouvoir les exploiter. Nous travaillons dans ce projet avec des données textuelles (résumés PubMed) et des bases de connaissances lexicales (LocusLink, OMIM, Gene Ontology).

2.1 PubMed

PubMed est un index des articles publiés en biologie : nous nous intéresserons en particulier à l'information contenue dans les résumés (abstracts) de ces articles.

2.2 LocusLink

LocusLink est une liste de gènes, mentionnant entre autres le nom officiel et les différents synonymes de chaque gène.

Nous nous intéresserons principalement aux champs suivants :

```
>>3638
LOCUS_TYPE: gene with protein product, function known or inferred
ORGANISM: Homo sapiens
PRODUCT: insulin induced gene 1 isoform 1
PRODUCT: insulin induced gene 1 isoform 2
PRODUCT: insulin induced gene 1 isoform 3
OFFICIAL_SYMBOL: INSIG1
OFFICIAL_GENE_NAME: insulin induced gene 1
ALIAS_SYMBOL: CL-6
ALIAS_SYMBOL: MGC1405
PREFERRED_PRODUCT: insulin induced gene 1 isoform 1
PREFERRED_PRODUCT: insulin induced gene 1 isoform 2
PREFERRED_PRODUCT: insulin induced gene 1 isoform 3
OMIM: 602055
```

2.3 OMIM

OMIM est une base de données contenant d'autres informations sur les gènes : fonction, rôle dans différentes pathologies, voies métaboliques, éventuelles interactions avec d'autres protéines, etc. Voir l'appendice A.3 pour un exemple.

2.4 Interlude : la notion d'ontologie

Les ontologies sont les concepts propres à un domaine (par exemple : la biologie moléculaire).

Un concept peut se décrire par un ensemble de mots synonymes – mais, pour compliquer les choses, un même mot peut avoir plusieurs sens, et donc appartenir à plusieurs ontologies.

Les ontologies sont reliées par diverses relations : par exemple des relations de généralisation (la relation "is a" (en français, "est un") de la programmation orientée objet) :

```
Objet
  |--Objet non physique
  `--Objet physique
       |--Objet vivant
       `--Objet non vivant
            `--Moyen de transport
                 |--Voiture
                 |    `--Voiture de sport
                 |--Bus
                 `--Train
...
```

En biologie, les familles de gènes ou de protéines sont des ontologies : on a donc des ontologies incluses les unes dans les autres et on peut représenter ces relations par un arbre. C'est très proche de la notion de taxonomie. (En littérature, on pourrait aussi penser à la notion de thésaurus : la relation est l'association d'idée.)

Voici un autre exemple, avec la relation "partie de".

```
Voiture
  |--Roue
  |    |--Frein
  |    `--Pneu
  `--Porte
       |--Vitre de la portière
       `--Poignée
...
```

Ce n'est pas toujours un arbre. Ainsi, en bioanalyse, on est souvent amené à utiliser successivement plusieurs logiciels ou bases de données : leur combinaison forme une "marche à suite" si la méthodologie est destinée à un être humain, un programme si elle est destinée à un ordinateur – nous parlerons de méthodologie. On peut par exemple concevoir une ontologie pour l'annotation des gènes d'un organisme nouvellement séquencé. Une telle méthodologie est une ontologie, ses différentes étapes sont elles-même des ontologies reliées par la relation "à faire avant" : on obtient un graphe orienté.

Un point important de la notion d'ontologie est qu'elles peuvent être utilisées par des ordinateurs. La principale application se trouve dans les moteurs de recherche : un moteur de recherche intelligent sera capable d'utiliser l'ontologie pour affiner ou étendre une recherche, en remplaçant un mot par des synonymes, des hyperonymes ou des hyponymes [10].

2.5 Gene Ontology

La Gene Ontology regroupe le vocabulaire utilisé pour décrire les gènes, aves les deux relations "est un" et "est une partie de" : en particulier, le vocabulaire décrivant la fonction et la localisation des protéines. Nous ne l'avons pas utilisée, mais une version ultérieure du logiciel pourra y recourrir afin d'affiner les recherches.

3 Algorithmes

Notre but est d'extraire des informations concernant les interactions entre gènes à partir des résumés de PubMed. Pour cela, il nous faut différencier les phrases qui décrivent des interactions de celles qui n'en décrivent pas. Nous présentons quelques méthodes se basant sur le principe suivant : on ne rencontre pas les mêmes mots dans ces deux classes de phrases. Nous détaillons le classificateur bayesien naïf [14] et l'indice IVI [30], que nous avons implémentés, et mentionnons, plus brièvement, les arbres de décision et les machines à vecteur support (SVM), un peu plus complexes.

3.1 Classificateur Bayesien naïf

Considérons l'exemple suivant : à l'aide d'un corpus annoté, on a constaté que :
- 40% des phrases décrivant une interaction contenaient le mot "downwards" contre 1% des phrases ne décrivant aucune interaction
- 20% des phrases décrivant une interaction contenaient le mot "act" contre 10% des phrases n'en décrivant pas
- 20% des phrases décrivaient des interactions.

On nous présente une phrase contenant le mot "downwards" et pas le mot "act" : quelle est la probabilité pour qu'elle décrive une interaction ?

Il s'agit d'un exercice assez simple sur les probabilités conditionnelles.

$$\begin{aligned} p &= P(\texttt{interaction}|\texttt{downwards} \text{ et pas } \texttt{act}) \\ &= \frac{P(\texttt{interaction} \text{ et } \texttt{downwards} \text{ et pas } \texttt{act})}{P(\texttt{downwards} \text{ et pas } \texttt{act})} \\ &= \frac{P(\texttt{downwards} \text{ et pas } \texttt{act}|\texttt{interaction})P(\texttt{interaction})}{P(\texttt{downwards} \text{ et pas } \texttt{act})} \end{aligned}$$

Mais si on suppose que la présence de "downwards" est indépendante de la présence de "act", on a

$$p = \frac{P(\texttt{downwards}|\texttt{interaction})P(\text{pas } \texttt{act}|\texttt{interaction})P(\texttt{interaction})}{P(\texttt{downwards} \text{ et pas } \texttt{act})}$$

Avec nos valeurs numériques :

$$p = \frac{0.4 \times 0.8 \times 0.2}{\dots} = \frac{0.064}{\dots}$$

De même pour la probabilité que ce ne soit pas une interaction :

$$1 - p = \frac{0.01 \times 0.1 \times 0.8}{\dots} = \frac{0.008}{\dots}$$

Comme les dénominateurs sont les mêmes dans les deux cas, on n'a pas besoin de les calculer, et on conclut que la phrase décrit probablement une interaction.

Remarque : on a dû supposer que la présence des mots "downwards" et "act" étaient indépendants – c'est éhontément faux, mais ça n'aura que peu de conséquences [8]. C'est là le sens du mot "naïf" dans l'expression "classificateur bayesien naïf".

On peut généraliser cela avec, non pas deux mots, mais quelques milliers. On fixe ensuite un seuil (par exemple 0.5) : si la probabilité est en dessous de ce seuil, on décrète que la phrase ne décrit pas d'interaction, si elle est au dessus, on décrète qu'elle en décrit une.

On constate que (malgré le défaut d'indépendance) ça marche très bien.

3.2 IVI (Indice de vraissemblance d'interaction)

Imaginons que nous n'ayons aucune connaissance en probabilités et que nous soyons donc incapables d'effectuer le raisonnement précédent. Reprenons un exemple comparable : le mot "downwards" apparait 10 fois dont 9 dans des phrases décrivant des interactions ; le mot "act" apparait 30 fois dont 3 dans des phrases décrivant des interactions.

On peut calculer

$$\mathrm{IVI} = \frac{9-1}{10} + \frac{3-27}{30} = -0.07.$$

(Chaque terme est le quotient de la différence entre le nombre d'occurences du mot dans les phrases décrivant des interactions et le nombre d'occurrences dans des phrases ne décrivant pas d'interactions et du nombre total d'occurrences de ce mot.)

Comme c'est négatif, on décrète que la phrase ne décrit pas d'interaction.

De manière plus générale, on considèrerait tous les mots qui apparaissent dans la phrase et pas juste ces deux-là.

3.3 Comparaison

L'indice IVI et les classificateurs bayesiens naïfs partent de points de vue complètements opposés : pour IVI, les données de départ sont les probabilités $P(\text{interaction}|\texttt{downwards})$ alors que pour les classificateurs bayesiens, c'est $P(\texttt{downwards}|\text{interaction})$. En effet, on peut écrire l'IVI en termes de probabilités :

$$\begin{aligned}\mathrm{IVI} = {} & P(\text{interaction}|\texttt{downwards}) - P(\text{pas interaction}|\texttt{downwards}) \\ & + P(\text{interaction}|\texttt{act}) - P(\text{pas interaction}|\texttt{act}).\end{aligned}$$

On peut s'attendre à ce que l'indice IVI d'une phrase soit d'autant plus grand (positif ou négatif) que la phrase est longue.

3.4 Autres algorithmes : arbres de décision

Dans les deux méthodes précédentes, on considérait les mots indépendemment les uns des autres. Les arbres de décision vont plus loin et tiennent compte d'éventuelles interactions (le défaut d'indépendance précédemment mentionné entre la présence de "act" et "downwards").

On commence par chercher le terme qui permet de différencier le plus les phrases décrivant une interaction de celles n'en décrivant pas. Par exemple, si on constate (sur le corpus d'apprentissage, annoté) que le mot "downwards" apparait dans 90% des phrases décrivant une interaction et que les autres mots ne font pas mieux (ni plus de 90% ni moins de 10%), on le retient.

On sépare alors le corpus d'apprentissage en deux morceaux : les phrases contenant le mot "downwards" et celles ne le contenant pas. On refait alors la même chose pour chacun de ces deux nouveaux corpus.

On continue ainsi et on obtient finalement un arbre :

Si "downwards" est présent
Alors
    Si "act" est présent
    Alors c'est une interaction
    Sinon ce n'est pas une interaction
Sinon
    Si "upwards" est présent
    Alors c'est une interaction
    Sinon ce n'est pas une interaction

Cette méthode pose quelques problèmes. D'une part, les corpus sur lesquels on travaille sont de plus en plus petits : pour estimer fidèlement les interactions entre les termes, il faut beaucoup de données – or le corpus annoté utilisé lors de la phase d'apprentissage est réalisé à la main : il est nécessairement limité.

D'autre part, ce manque de données risque de conduire à une surmodélisation des données d'apprentissage : l'arbre fonctionnera extrèmement bien sur ces données, mais si on lui donne de nouvelles données, les résultats seront médiocres. L'arbre est incapable d'extrapoler. Pour contourner ce problème, il est nécessaire d'élaguer l'arbre.

Enfin, les méthodes de prédiction basées sur des arbres sont instables : si on a des données d'apprentissage très légèrement différentes (par exemple, un individu en moins), on obtient un arbre complètement différent. Pour contourner ce problème, on peut rééchantilloner (en anglais : bootstrap) les données d'apprentissage et construire un arbre pour chacun de ces échantillons. On obtient alors une forêt d'arbres de prédiction (en fait, cette méthode pour stabiliser des prédicteurs est générale : en anglais ça s'appelle le bagging). Face à une nouvelle phrase, ces différents arbres feront des prédictions différentes : on pourra par exemple retenir l'avis majoritaire.

3.5 Autres algorithmes : SVM

Pour tout ce qui est régression (prédiction d'une variable, quantitative ou qualitative, à l'aide d'autres variables, quantitatives ou qualitatives) avec plus de variables que d'individus (ici, plus de mots que de textes), les SVM (Support Vector Machines) donnent généralement de très bons résultats.

Si vous êtes amené à faire du Data Mining ou de l'Analyse de Données par la suite, vous rencontrerez à nouveau les SVM.

4 Choix technologiques

En quel langage implémenter ces algorithmes ? Faut-il se tourner vers des langages spécialisés dans les calculs scientifiques ou dans l'apprentissage automatique ? Faut-il se tourner vers des logiciels (et non plus des langages) de statistique ou de Text Mining ? Faut-il se tourner vers un langage compilé, d'exécution rapide, comme le C ou le C++ ? Faut-il préférer un langage à la mode, comme Java ? Ou au contraire un langage interprété, plus lent à l'exécution, mais qui permet un développement plus rapide ? Et quel langage interprété préférer ? Perl ? Python ? Un langage moins répandu ?

Nous présentons les différentes possibilités que nous avons examinées, en terminant par Python, que nous avons retenu.

4.1 Logiciels de statistiques

Nous avons assez rapidement rejeté les logiciels d'analyse de données (Weka, gdmdtree, etc.) ne permettant pas de programmer, car ils nous enferment dans les algorithmes ou méthodes déjà implémentées [43].

Nous avons aussi rejeté les langages orientés vers le calcul statistique (R), car ils sont peu adaptés à la manipulation du texte [33].

4.2 Perl

Les avantages de Perl sont les suivants : d'une part, il est plus utilisé que python, d'autre part, les expressions régulières permettent d'écrire très rapidement des programmes compacts qui manipulent ou découpent efficacement du texte. On peut aussi mentionner le nombre très élevé de modules, pour réaliser les tâches les plus diverses, par exemple de la bioinformatique : le projet BioPerl permet ainsi d'automatiser la plupart des tâches de bioanalyse [20].

Le principal inconvénient de Perl est le suivant : on peut programmer comme on veut. Si on veut programmer proprement et si on est capable de s'imposer cette discipline, Perl vaut n'importe quel autre langage. Par contre, pour un public qui vient d'apprendre à programmer, c'est l'un des langages les pires.

Pour cette raison et malgré son utilisation très fréquente en bioinformatique, nous avons rejeté Perl.

4.3 Python

Les avantages de Python sont les suivants.

C'est un langage beaucoup plus propre que Perl : il impose un style de programmation uniforme et facilite ainsi la la collaboration de plusieurs programmeurs, même si certains d'entre eux sont peu expérimentés. Python est d'ailleurs souvent utilisé lors de l'apprentissage de la programmation à des non-informaticiens ou à des lycéens [39] ; ainsi, la bibliothèque NLTK a été développée pour un cours d'informatique appliquée à la linguistique [26].

Python est aussi utilisé en physique, pour des simulations numériques (on pourrait faire la même chose en Perl, avec le module PDL, mais c'est moins utilisé) et en bioinformatique : BioPython permet d'automatiser des tâches courantes de bioanalyse – après avoir établit une "procédure" pour une certaine tâche (disons, l'annotation des gènes), on peut expliquer cette procédure à un ordinateur – BioPython présente l'avantage d'être très simple à utiliser, et donc d'être utilisable directement par le biologiste concepteur de la procédure, sans qu'il faille de réelles compétences en programmation [36].

C'est aussi un outil de RAD (Rapid Application Development), avec par exemple Boa Constructor [3] et, dans le domaine du développement Web, le couple Zope/Plone est en passe de devenir un concurrent sérieux de Php [51, 31, 49].

Python est très utilisé autour de Java, car il existe un interpréteur python en java : cela permet de débugguer facilement des applications java ou de proposer au client de les configurer lui-même à l'aide d'un vrai langage de programmation, facile à utiliser – Java est avant tout un argument "marketting" [18, 34, 27].

4.4 NLTK

Nous avons choisi la bibliothèque NLTK pour manipuler les textes sous Python : elle nous permet facilement de découper les textes en phrases, en mots, de lemmatiser les mots, etc. [26, 35].

5 Implémentation

5.1 Présentation de l'algorithme

L'algorithme développé (figure 1) ici comporte deux parties distinctes :

- un module d'apprentissage qui permet de générer des informations essentielles à l'analyse ultérieure des données textuelles ; il contient un calcul d'un indice d'interaction et crée un lexique de gènes.
- un module d'utilisation qui va transformer l'information biologique textuelle en une base de données, facilement interrogeable.

Le calcul de l'indice d'interaction (IVI et Bayes) s'appuie le corpus d'apprentissage c'est à dire un classement de phrases décrivant ou non des interactions (expertise humaine sur 250 résumés PubMed). À partir de ce corpus, le programme va générer une liste de mots et calculer pour chacun d'eux une "probabilité d'interaction". D'autre part le programme va créer un lexique de gènes à partir de la base de connaissances LocusLink. Ce lexique contient le nom officiel du gène, ses synonymes, son symbole officiel ainsi que d'autres informations utiles comme le numéro OMIM ou encore le numéro de l'entrée LocusLink.

Pour le module d'utilisation, le progamme va récupérer l'information textuelle sous forme HTML, en stocker le numéro et traiter le résumé de la publication. Le texte va être découpé en phrases puis en mots, et on ne retiendra que les phrases comportant au moins deux noms de gènes. En utilisant les données du module d'apprentissage, on affectera une probabilité d'interaction pour chaque phrase en additionnant les indices de chaque mot de la phrase.

Les données extraites sont exportées sous forme de requêtes SQL prêtes à être insérées dans la base de données prévue à cet effet.

5.2 Objets

Pour faciliter la répartition des tâches et une éventuelle reprise du projet par d'autres, nous avons choisi de séparer le programme en différentes classes : Phrase (une liste de mots ou de lemmes), Texte (une liste de Phrases), Classificateur (classe abstraite qui décrit l'interface des différents classificateurs : une phase d'apprentissage avec un corpus d'apprentissage, une méthode pour calculer la "probabilité" d'une phrase), IVI, Bayes (deux implémentations de cette interface), LocusLink (un dictionnaire contenant les noms officiels et les synonymes). Une fois ces classes écrites, le corps du programme est très réduit.

5.3 Découpage des textes en phrases et des phrases en mots

Ça a l'air vraiment trivial, mais en fait, c'est un peu délicat à programmer correctement : on dispose de textes et on cherche à les découper en phrases, puis à découper les phrases en mots.

Pour le découpage en phrases, on commence par essayer de découper le texte à chaque point. Mais on peut trouver des points en plein milieu d'une phrase, en particulier dans des expressions numériques. On serait tenté de dire qu'une phrase commence par une lettre majuscule : mais c'est faux, car une phrase peut commencer par un nom de gène, qui peut être partiellement en minuscules. On peut décréter qu'une phrase se termine par un point suivi d'un espace – on se débarasse ainsi du point décimal. Mais il reste encore un problème : on trouve parfois des initiales suivies d'un point. On va donc décréter qu'une phrase se termine par un point non précédé d'une majuscule.

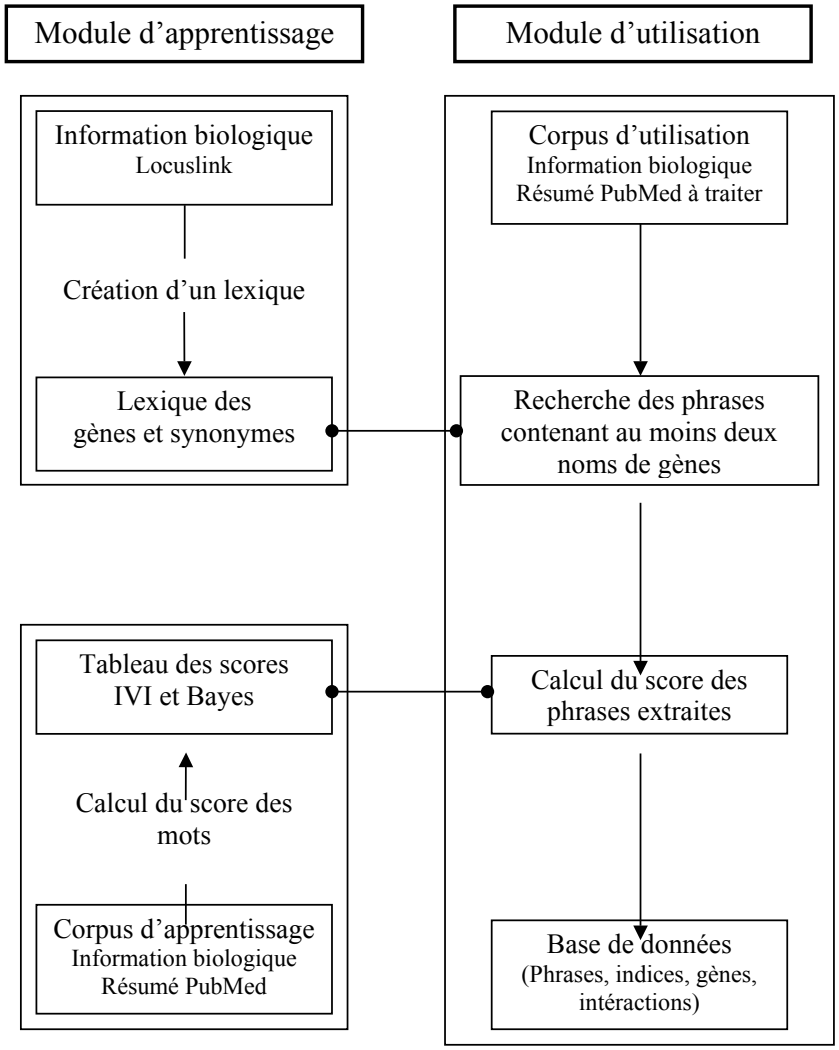

FIG. 1 – Déroulement de l'algorithme

On peut expliquer ça à l'ordinateur avec une *expression régulière*.

Pour le découpage en mots, c'est plus simple : les mots sont séparés par des espaces, avec parfois un symbole de ponctuation (point, virgule, parenthèse, guillemet, etc.) collé au mot. Ici encore, on utilise une expression régulière.

5.4 Corpus d'apprentissage

Il a été fait à la main : on demande au programme d'afficher les phrases qui contiennent au moins deux noms de gènes, on lit les phrases une par une, et on les répartit (à la main) dans deux fichiers, selon si elles décrivent des interactions ou non.

5.5 Indice et probabilité

Les deux méthodes que nous avons retenues, le classificateur bayesien et l'indice IVI, donnent des résultats de nature différente : dans un cas, une probabilité, *i.e.*, un nombre entre 0 et 1 avec une valeur seuil en 1/2, dans l'autre, un nombre quelconque, avec une valeur seuil en 0. Pour que l'utilisateur puisse les utiliser et les comparer facilement, nous allons transformer l'indice IVI en quelque chose qui ressemble à une probabilité (figure 2).

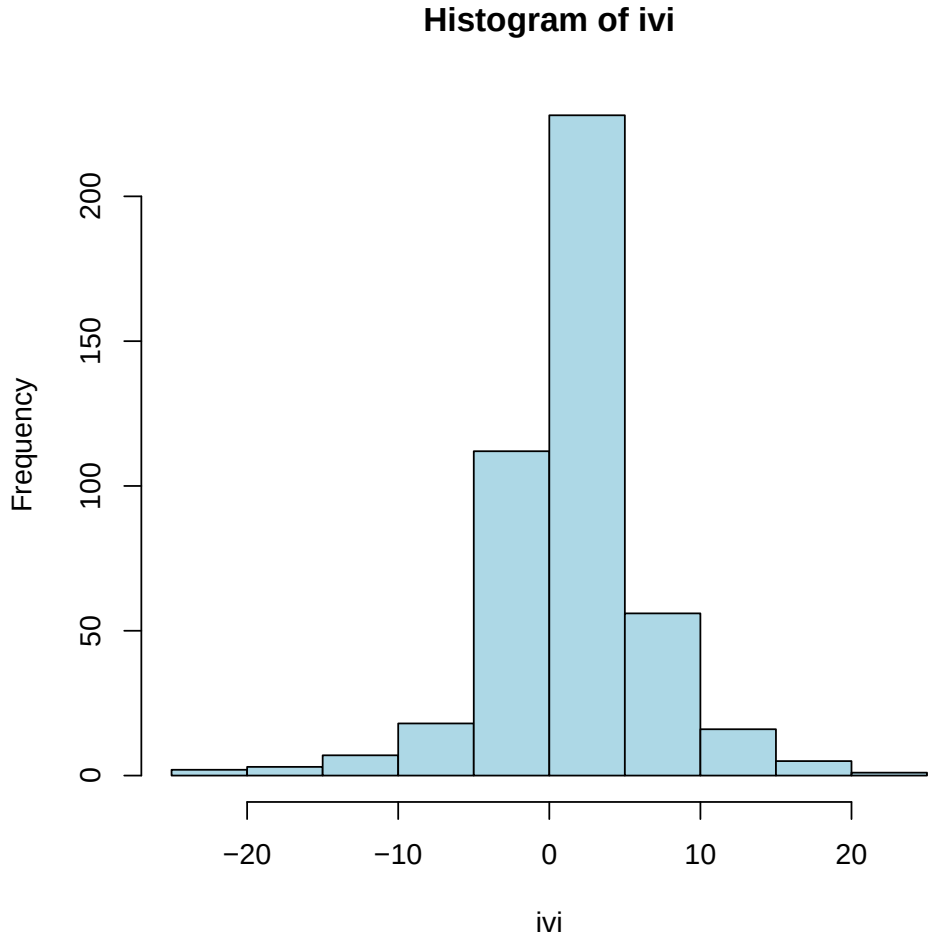

Fig. 2 – Histogramme des scores IVI

En apprentissage automatique (Machine Learning) et plus particulièrement dans les réseaux de neurones, on utilise souvent la "fonction sigmoïde" pour passer d'un nombre réel quelconque à une probabilité (figure 3).

$$\sigma_k : \begin{cases} \mathbf{R} & \longrightarrow & [0,1] \\ x & \longmapsto & \dfrac{1}{1+e^{-kx}} \end{cases}$$

Pour trouver une "bonne" valeur pour le coefficient k de la fonction sigmoïde, on peut regarder la forme de l'histogramme pour différentes valeurs et choisir, de manière empirique, la "meilleure" valeur. On trouve $k = 1/3$

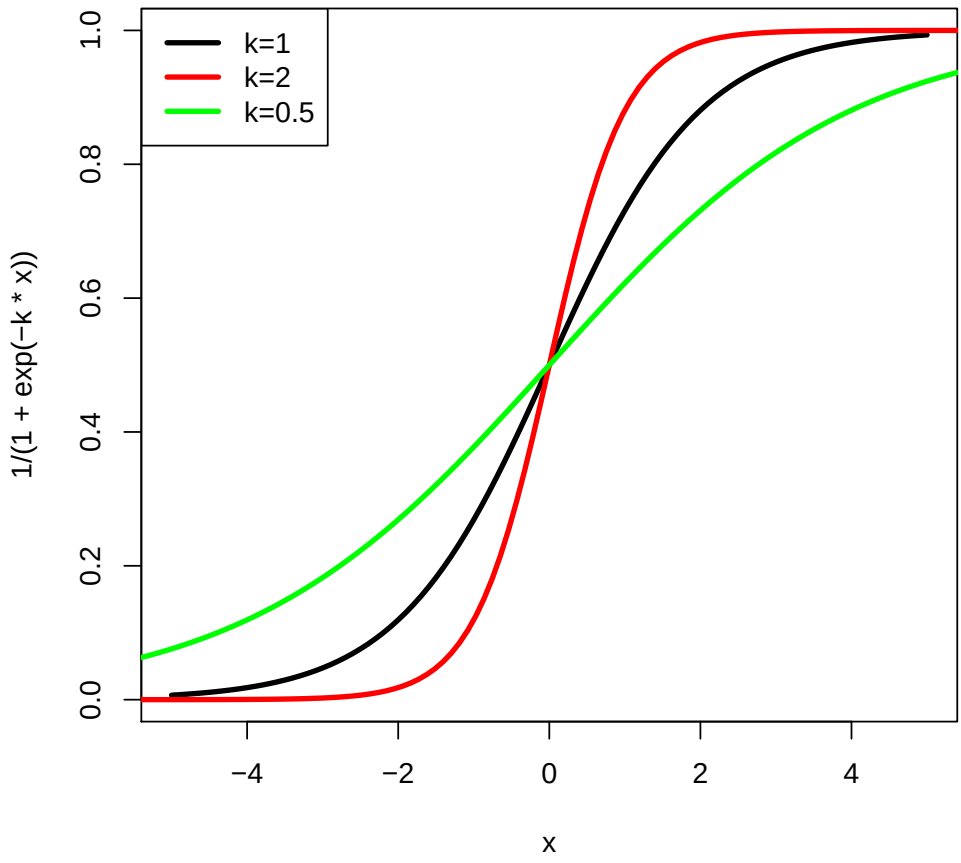

FIG. 3 – La fonction sigmoïde

(figure 4).

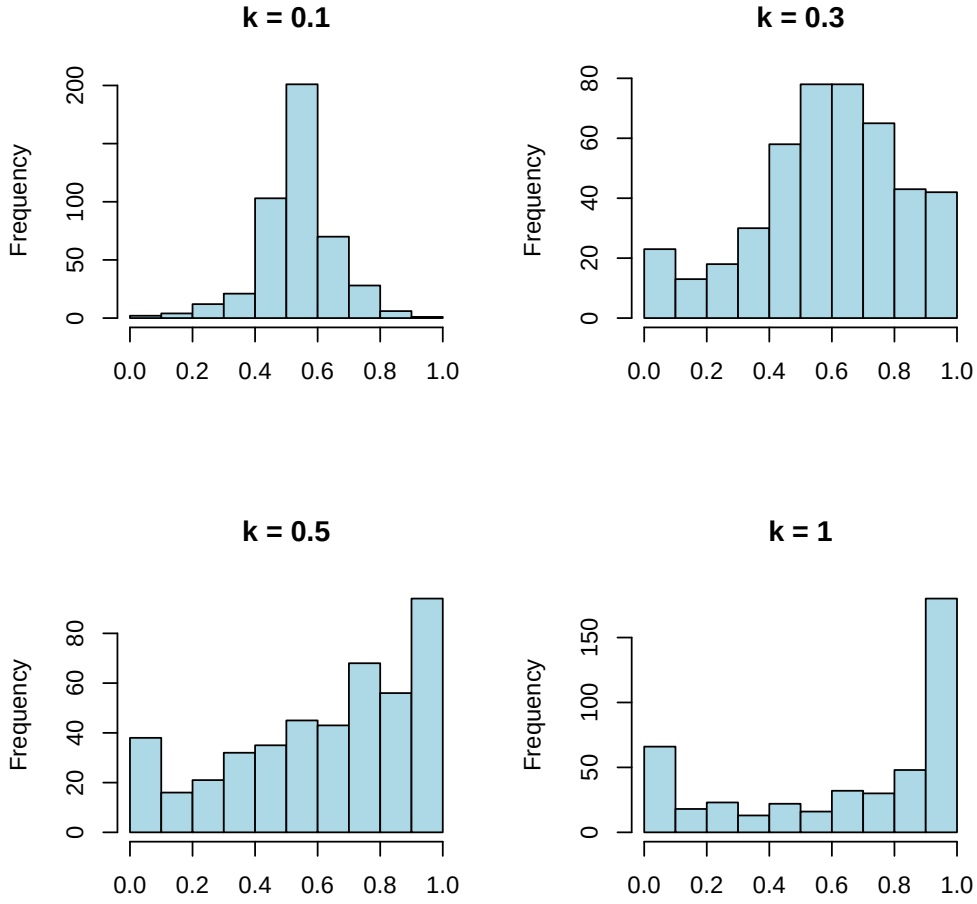

FIG. 4 – Tatonnements pour trouver le paramètre de la fonction sigmoïde qui donne la répartition la plus uniforme des "pseudo-probabilités" d'IVI. Pour $k = 0.1$, les valeurs sont très concentrées autour de 0.5 : ça ne convient pas. Pour $k = 1$, les valeurs sont très concentrées autour de 0 et de 1 : cela permet d'avoir une réponse binaire (oui/non), mais pas vraiment d'estimer la qualité de la réponse. Par contre, pour $k = 0.3$ ou $k = 0.5$, les valeurs sont mieux réparties. De manière tout à fait arbitraire, on choisit $k = 1/3$.

Une autre manière de trouver la valeur de k consiste à comparer la valeur de l'indice IVI avec la probabilité donnée par le classificateur bayesien et à réaliser une régression non linéaire : on trouve $k = 1.2$.

6 Résultats, conclusions, limitations, perspectives

Après avoir présenté et évalué nos résultas, nous présentons quelques améliorations possibles de notre travail – le manque de temps ne nous a pas permis de nous y attaquer.

6.1 Résultats

Pour évaluer la qualité des résultats, nous avons coupé le corpus d'apprentissage en deux, utilisé la première partie pour l'apprentissage des classificateurs et le reste comme échantillon de test.

Voici le nombre de phrases qui ont été bien ou mal classées.

Bayes	interaction prédite	pas d'interaction prédite
interaction	100	14
pas d'interaction	59	83

IVI	interaction prédite	pas d'interaction prédite
interaction	87	27
pas d'interaction	59	83

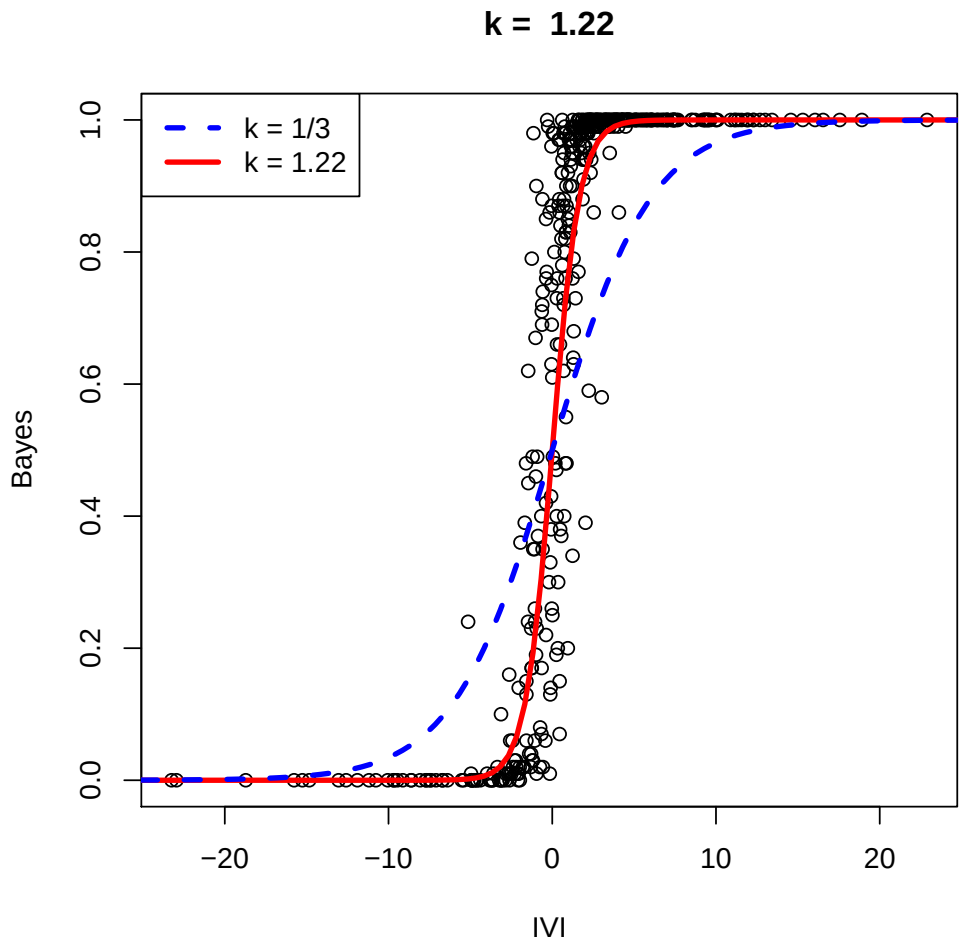

FIG. 5 – Calcul du paramètre de la fonction sigmoïde en comparant l'indice IVI et la probabilité donnée par le classificateur de Bayes à l'aide d'une régression non-linéaire.

On en déduit le *taux de rappel*, *i.e.*, le pourcentage de phrases trouvées parmi les phrases décrivant des interactions, et le *taux de précision*, *i.e.*, le pourcentage de phrases qui décrivent effectivement des interactions parmi les phrases trouvées :

Méthode	Rappel	Précision
Bayes	88%	63%
IVI	76%	60%

(Sans lemmatisation, les chiffres sont à peu près les mêmes : identiques pour IVI, un point de moins pour le classificateur de Bayes.)

Un corpus d'apprentissage plus gros et de meilleure qualité permettrait peut-être d'améliorer ces chiffres.

6.2 Automatisation de la construction du corpus

Actuellement, la base n'a été construite que sur un corpus de quelques centaines de résumés d'articles, résultats d'une recherche sur PubMed, récupérés à la main (sous la forme d'un gros fichier HTML). On pourrait automatiser cela, d'une part en prenant une copie complète de PubMed puis, toutes les semaines, en allant chercher automatiquement les nouveaux articles.

6.3 Découpage ou étiquetage

Une prochaine version du programme pourrait remplacer le découpage en mots des phrases par un étiquetage (ou des annotations) – c'est d'ailleurs ce que fait NLTK. Cela permettrait de faciliter la reconnaissance des noms de gènes qui comportent plusieurs mots, des locutions qui comportent plusieurs mots (exemple : *de novo*) et de gérer certaines ambiguïtés (comme le caractère "/", qui apparait parfois en plein milieu d'un nom de gène, parfois entre deux noms de gènes).

> *Here we show that LXRalpha/RXR and LXRbeta/RXR induce SR-BI transcription in human and murine hepatoma cell lines, and in 3T3-L1 preadipocytes independently of SREBP-1.*

Cela permettrait aussi d'étiqueter les syntagmes[1], première étape vers un réel traitement *linguistique* des résumés.

6.4 Desambiguations

Un même mot est parfois utilisé avec des sens très variés : ainsi, certains noms de gènes sont aussi des noms communs ou des adjectifs. Il devrait être possible de distinguer les deux en remarquant qu'ils ne sont pas utilisés dans le même contexte. Ici, il peut s'agir du contexte sémantique (le sens général du texte environnant, que l'on peut trouver en manipulant des "sacs de mots") ou grammatical (un nom de gène s'utilise comme un nom, pas comme un adjectif).

Même si c'est bien le gène qui apparaît dans le texte, il peut être accompagné d'autres mots qui en modifient complètement le sens. Ainsi, dans notre corpus d'apprentissage, l'expression "LXR agonist" revenait très souvent. Il s'agit en fait de reconnaître les syntagmes.

Il y a aussi des ambiguités dans l'autre sens : certains noms de gènes ne sont pas reconnus, car ils ne sont pas dans LocusLink.

symbole utilisé	symbole le plus proche dans LocusLink	différence
"apoE"	"APOE"	casse
"PPARgamma"	"PPAR gamma"	perte d'un espace
"LXR beta"	"LXRbeta"	ajout d'un espace
"liver X receptor"	"liver X receptor, alpha"	préfixe

Une prochaine version du programme pourrait tenter de reconnaître ces mots, en remarquant qu'ils ne sont pas dans le dictionnaire (par exemple, WordNet) et qu'ils sont proches (identiques aux espaces et à la casse près, préfixes) de symboles de LocusLink.

6.5 Transformations linguistiques

Une prochaine version du programme pourrait effectuer des transformations linguistiques sur les phrases, afin de les simplifier. Par exemple, une transformation passif/actif

[1] Un syntagme, c'est un groupe de mots.

Avant : *La souris a été mangée par le chat.*
Après : *Le chat a mangé la souris.*

Il peut aussi s'agir de transformation d'une phrase complexe en plusieurs phrases simples.

Avant : *Le chat a mangé la souris et cassé le vase.*
Après : *Le chat a mangé la souris. Le chat a cassé le vase.*

6.6 Reconnaissance de motifs

Dans ce projet, nous nous sommes limités à la *recherche d'information* dans les textes (trouver les phrases qui contiennent l'information qui nous intéresse) et n'avons que peu abordé l'*extraction d'information* (trouver la nature de l'interaction, positive ou négative, le sens de l'interaction, etc.).

Une manière de procéder consisterait à établir une liste de "motifs", *i.e.*, de structures grammaticales de phrase, décrivant les interactions recherchées. Comme les phrases effectivement utilisées dans les résumés peuvent être très complexes, l'étape mentionnée précédemment, les transformations et simplifications des phrases, est primordiale.

7 Conclusion

Ce projet s'inscrit dans le vaste domaine de l'extraction de données (data mining) et en particulier de la fouille de texte (text mining). Le logiciel proposé permet d'extraire à partir des résumés une information qui devra cependant être validée par le biologiste. En effet, le résultat d'une recherche peut être dilué par des faux positifs (phrases extraites alors qu'elles ne décrivent pas d'intéraction : 40%). Par contre il y a peu de perte d'information puisque moins de 10% des phrases décrivant une intéraction ne sont pas extraites. Ce logiciel est donc largement utilisable ; il permet d'extraire rapidement une information qui aurait nécessité plusieurs heures voire plusieurs jours de lecture. La maquette fournie à l'issue de ce projet laisse la place à des améliorations : notamment l'ajout d'outils linguistiques et le recours à des ontologies (Gene Ontology) en plus des outils statistiques afin d'affiner les résultats.

Ce projet nous a apporté une vision plus précise de l'apport de la bioinformatique dans le traitement des données biologiques et en particulier des données textuelles. Un point fort de cette réalisation a été l'échange de nos expériences théoriques et pratiques, ce qui nous a permis de concrétiser une partie de l'enseignement reçu au cours de ces six derniers mois.

Références

[1] *Bayesien network classifiers*, `http://www.vision.ethz.ch/gm/bnclassifiers.pdf`, 2002.
Les classificateurs bayesiens naïf supposent que les variables prédictives sdont indépendantes. Si ce n'est pas le cas, on peut essayer de trouver un modèle moins trivial, comme des arbres bayesiens ou des réseaux bayesiens (c'est important, les réseaux bayesiens : avant quelques années, ils deviendront aussi répandus que les réseaux de neurones). On qualifie de "modèle graphique" une telle description graphique des liens de dépendance entre les variables (ça aussi, c'est à la mode).

[2] *BioText*, `http://biotext.berkeley.edu/`.

[3] *Boa constructor*, `http://boa-constructor.sourceforge.net/`.
Un outil de RAD (Rapid Application Development) pour Python. Lire aussi l'article de DevChannel, *Rapid Application Developments*, Michael 'STIBS' Stibane, février 2004, `http://www.devchannel.org/devtoolschannel/04/02/02/1939223.shtml`.

[4] E. Brill, *Etiqueteur de brill*, `http://www.cs.jhu.edu/~brill/RBT1_14.tar.Z`.
C'est un étiqueteur lexical : pour chaque mot d'un texte, il en donne la nature grammaticale (Part Of Speech, pos). Le principe de l'algorithme est le suivant : le programme connait, pour chaque mot, toutes les natures grammaticales possibles ; il commence par assigner à chaque mot la nature la plus probable ; il les corrige ensuite en cherchant des motifs dans un texte (par exemple, on ne peut pas trouver toutes les natures possibles après une préposition). Ces règles sont apprises automatiquement à partir d'un corpus d'apprentissage, étiqueté à la main.

[5] *Caderige*, `http://caderige.imag.fr/Rapport.html`.
Projet d'extraction d'informations concernant des interactions entre gènes à partir de données bibliographiques (MedLine)

[6] M. Crocker and F. Keller, *Connectionist and statistical language processing*, `http://www.cogsci.ed.ac.uk/~keller/teaching/connectionism/lecture10_4up%.pdf`, 2001.
Notes d'un cours d'apprentissage automatique (Machine Learning), avec des exercices.

[7] G. de Chalendar, *Svetlan, un système de structuration du lexique guidé par la détermination automatique du contexte thématiquecontexte thématique*, `http://www.limsi.fr/Individu/gael/ManuscritThese/HTML/index.html`, 2001.
Svetlan est un logiciel permettant de trouver la classe sémantique des mots à l'aide de leur contexte.

[8] P. Domingos and M. Pazzani, *On the optimality of the simple bayesian classifier under zero-one loss*, http://www.cs.washington.edu/homes/pedrod/mlj97.ps.gz, 1997.
On sait que le classificateur bayesien naïf est optimal si les données sont indépendantes, mais on l'utilise aussi souvent, sans aucun problème, alors que cette hypothèse n'est pas du tout vérifiée. Cet article regarde à quel point on peut s'affranchir de cette hypothèse.

[9] M. Grobelnik et al., *Text mining as an integration of several related research areas : report on kdd2000 workshop on text mining*, SIG Explorations, **2** (2), http://www.acm.org/sigs/sigkdd/explorations/issue2-2/KDD2000TextWkshp.p%df, 1999.

[10] R.A. Calogero et al., *Mining litterature to improve biological knowledge extraction by microarray transcriptional profiling*, http://www.bioinformatica.unito.it/complex_systems/medmole2002-V2.ppt.
Présentation de MedMOLE

[11] Y. Regev et al., *Rule-based extraction of experimental evidence in the biomedical domain the kdd cup 2002 (task 1)*, KDD Explorations, **4** (2), p. 90–92, http://www.acm.org/sigs/sigkdd/explorations/issue4-2/regev.pdf.
Présentation du travail de l'équipe gagnante du concours KDD Cup 2002. Ils recherchent des motifs dans des textes : ces motifs peuvent être appris automatiquement par l'ordinateur (modèles de Markov cachés) ou conçus à la main – ils ont choisi cette deuxième approche. Pour cela, le texte est préalablement étiqueté (POS tagger), les mots sont regrouppés en syntagmes, et on compare ces syntagmes avec les motifs préalablement définis.

[12] *GATE, a general architecture for text engineering*, http://gate.ac.uk/.
Une bibliothèque (et même un "framework") en Java, libre (LGPL), pour la linguistique. Pour un projet sérieux, de grande envergure, c'est un bon choix. Mais il faut probablement une connaissance préalable en Java, XML, Ant, peut-être même Swing, et un certain temps d'apprentissage.

[13] M. Goebel and L. Gruenwald, *A survey of data mining and knowledge discovery software tools*, SIGKDD Explorations, **1** (1), p. 20–33, 1999.

[14] P. Graham, *A plan for spam*, http://www.paulgraham.com/spam.html, august 2002.
L'article qui a popularisé les classificateurs bayesiens pour la lutte contre le spam.

[15] S. Guharay, *Text categorization of south asian last names*, http://www.tartarus.org/~martin/PorterStemmer/index.html, july 2003.
Un exemple d'application des algorithmes de classification (SVM et classificateur bayesien) : trouver l'origine d'un patronyme à l'aide des n-grammes (suites de n lettres) qui le composent.

[16] *HUGO : the human genome organization*, http://www.gene.ucl.ac.uk/hugo/.

[17] A. Ingold, *Expérience de couplage entre bases de données factuelles et bases de données bibliographiques : identification dans medline des gènes décrits dans flybase et application à l'extraction d'informations sur les interactions génétiques ou moléculaires à partir de publications*, http://ingold.free.fr/statique/these/PhD-Chapitre-7.html#Heading5463, 2002.

[18] *Jython*, http://www.jython.org/.
Jython est un interpréteur Python écrit en Java, que l'on peut utiliser, par exemple, pour débugguer ou configurer un programme en Java.

[19] *KDD cup 2002*, http://www.biostat.wisc.edu/~craven/kddcup/, 2002.
La KDD Cup est un concours de DataMining, qui a lieu (à peu près) tous les ans. En 2002, c'était de la bioinformatique, avec la moitié du sujet sur la sélection d'articles de recherche.

[20] C. Letondal and K. Schuerer, *Bioperl course*, http://www.pasteur.fr/recherche/unites/sis/formation/bioperl/, 2003.

[21] *Lucene*, http://jakarta.apache.org/lucene/.
Outil de recherche textuelle, en Java, dans le projet Jakarta (Apache). Libre. C'est plus de la récupération d'information (Information Retrieval, IR) que de l'extraction d'information (Information Extraction, IE).

[22] A. McCallum and K. Nigam, *A comparison of event models for naive Bayes classification*, http://www-2.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf, 1998.
L'expression "classificateur bayesien naïf" recouvre deux notions différentes : un modèle dit "binomial", dans lequel on s'intéresse à la présence ou à l'absence des mots, et un modèle dit "multinomial", dans lequel on s'intéresse au nombre d'occurrences de chaque mot. Cet article souligne les différences entre ces deux modèles.

[23] A. K. McCallum, *Bow : A toolkit for statistical language modeling, text retrieval, classification and clustering*, 1996, http://www.cs.cmu.edu/~mccallum/bow.
Bow et Rainbow sont des bibliothèques en C pour la classification de documents (et justement, une partie de notre projet s'intéresse à la classification des phrases des résulés d'articles scientifiques).

[24] *MedMOLE*, http://www.bioinformatica.unito.it/bioinformatics/medmole/welcome.html.

[25] *MeSH (Medical Subject Headings)*, http://www.nlm.nih.gov/mesh/filelist.html.
Vocabulaire médical.

[26] *NLTK : Natural Language Toolkit*, http://nltk.sourceforge.net/tutorial/index.html.
Un module Python pour le traitement automatique des langues naturelles (TALN), utilisé dans certains cours d'informatique appliquée à la linguistique.

[27] U. Ogbuji, *Charming Jython : Learn how the Java implementation of Python can aid your development efforts*, http://www-106.ibm.com/developerworks/java/library/j-jython.html, may 2003.
Présentation de Python pour programmeurs Java.

[28] *Omim (online mendelian inheritance in man)*, http://www.ncbi.nlm.nih.gov/Omim/.
Catalogue de gènes.

[29] Toshihide Ono, Haretsugu Hishigaki, Akira Tanigami, and Toshihisa Takagi, *Automated extraction of information on protein-protein interactions from the biological literature*, http://binfo.ym.edu.tw/edu/seminars/pdf/110201.pdf.
Présentation de la problématique de recherche d'interactions entre protéines dans des articles de recherche, présentation de l'étiqueteur de Brill, présentation de l'algorithme de lemmatisation de de Porter, présentation de quelques transformations linguistiques permettant de passer d'une phrase complexe (par exemple une phrase décrivant plusieurs interactions) à plusieurs phrases simples (chacune décrivant une seule interaction).

[30] V. Pillet, *Méthodologie d'extraction automatique d'information etc.*, http://ms161u13.u-3mrs.fr/memoires/ViolainePillet_TS.pdf, janvier 2000, thèse de doctorat.
Définition de l'indice IVI (Indice de Vraissemblance d'Interaction).

[31] *Plone*, http://plone.org/.
Plone est un CMF (Content Management Framework), *i.e.*, un gestionnaire de contenu,

[32] M. Porter, *The Porter stemmer algorithm*, http://www.tartarus.org/~martin/PorterStemmer/index.html.
Présentation de quelques implémentations de l'algorithme de Porter, l'algorithme de lemmatisation le plus courrant, par son auteur.

[33] *The R project for statistical computing*, http://www.r-project.org/.
R est un logiciel et un langage de programmation de calculs statistiques, très utilisé en recherche, en finance, en bioinformatique. C'est un projet libre.

[34] N. Rappin, *Tips for scripting java with jython*, http://www.onjava.com/pub/a/onjava/2002/03/27/jython.html, march 2002.
Présentation de quelques utilisations possibles de Jython : débugguage et configuration d'un programme en Java.

[35] K. Russell, *Review : Software : Natural language toolkit (nltk) 1.2*, http://www.linguistlist.org/issues/14/14-3165.html, 2003, LINGUIST List 14.3165.
Critique (objective) de NLTK.

[36] K. Schuerer and C. Letondal, *Python course in bioinformatics*, http://www.pasteur.fr/recherche/unites/sis/formation/python/, 2002.

[37] H. Schütze, *Open source text mining*, http://www.cs.utk.edu/tmw03/keynote.ppt.

[38] *What is stemming ?*, http://www.comp.lancs.ac.uk/computing/research/stemming/general/index.h%tm.
Présentation de différents algorithmes de lemmatisation (en anglais : *stemming*), le plus répandu étant celui de Porter.

[39] G. Swinnen, *Apprendre à programmer avec python*, éditions O'Reilly, ISBN 2-84177-294-2, aussi disponible sur internet : http://www.ulg.ac.be/cifen/inforef/swi/python.htm, 2003.
Un cours d'introduction à la programmation, basé sur Python.

[40] *TMI : Text mining infrastructure*, http://hddi.cse.lehigh.edu/.

[41] J. Udell, *Working with bayesian categorizers*, http://www.xml.com/pub/a/2003/11/19/udell.html, november 2003.
Présentation de Bow/Rainbow (des bibliothèques de C pour manipuler des "sacs de mots") et de `AI::Categorizer` (idem, en Perl).

[42] Špela Vintar, Ljupco Todorovski, Daniel Sonntag, and Paul Buitelaar, *Evaluating context features for medical relation mining*, http://muchmore.dfki.de/pubs/ecml_final_new.pdf.
Lors de la recherche de relations entre des notions (par exemple, des interadctions entre des gènes ou des protéines) dans des textes, on constate qu'un même type de relation apparît souvent dans le même contexte, *i.e.*, entouré des mêmes mots : l'article utilise MeSH et des arbres de classification pour les identifier.

[43] *Weka, data mining software in java*, http://www.cs.waikato.ac.nz/ml/weka/, University of Waikato.
Weka est un outil graphique de fouille de données, écrit en Java.

[44] K. Williams, *Automatic document classification with perl*, `http://mathforum.org/~ken/bayes/bayes.html`.
Présentation du module Perl `AI::Categorize`

[45] _____, *An introduction to machine learning with Perl*, `http://conferences.oreillynet.com/presentations/bio2003/williams_ken.pp%t`, `http://www.campstaff.com/~ken/talks/MLPerl.pdf`, O'Reilly Bioinformatics Conference, february 2003.
Présentation de l'Apprentissage Automatique (ML : Machine Learning), utilisation de Perl à cet effet, présentation des SVM.

[46] P. Wolper, *Introduction à la calculabilité*, InterEdition, ISBN 2-7296-0372-7, 268 pages, 1991.
Langages régulier (ou rationnel), expressions régulières et automates finis sont différents aspects d'une même notion : ce livre les présente, de manière théorique (sans aucune ligne de code) mais très claire. Il parle aussi de complexité, de NP-complétude, de machine de Turing, etc.

[47] *WordNet, a lexical database for the English language*, `http://www.cogsci.princeton.edu/~wn/`.

[48] M.-H. Yang, *Gentle guide to support vector machines*, `http://vision.ai.uiuc.edu/mhyang/talk/svm-talk.pdf`.
Encore une présentation des SVM, un peu plus théoriques.

[49] V. Zoonekynd, *Zope : programmation web avec Python*, `http://zoonek2.free.fr/UNIX/34_Zope/zope.html`.

[50] _____, *Statistiques avec R*, `http://zoonek2.free.fr/UNIX/48_R/all.html`, 2004.
Version préliminaire d'un livre sur les possibilités graphiques du logiciel de statistiques R, dont nous nous sommes inspiré pour écrire la fonction permettant d'observer, de manière interactive, l'effet d'un changement du paramètre de la fonction sigmoïde sur la forme de l'histogramme des pseudo-probabilités d'IVI.

[51] *Zope*, `http://www.zope.org/`.
Zope est un "framework" pour faire de la programmation Web en Python.

[52] B. Zupan and J. Demsar, *Orange for beginers*, `http://magix.fri.uni-lj.si/orange/doc/ofb/`, november 2003.
Orange est un module Python pour l'apprentissage automatique (Machine Learning), qui propose, entre autres, des classificateurs bayesiens, des arbres de classification, les plus proches voisins, mais aussi le bagging et le boosting. Comparé à d'autres modules/bibliothèques du même genre, c'est *très* bien documenté. Les calculs ne sont pas implémentés en Python, c'est donc très rapide.

A Données

A.1 PubMed

Voici un exemple d'enregistrement PubMed.

```
PMID- 14720212
OWN - NLM
STAT- in-data-review
DA  - 20040114
IS  - 0022-3042
VI  - 88
IP  - 3
DP  - 2004 Feb
TI  - A liver X receptor and retinoid X receptor heterodimer mediates
      apolipoprotein E expression, secretion and cholesterol homeostasis in
      astrocytes.
PG  - 623-34
AB  - Apolipoprotein E (apoE) is an important protein involved in lipoprotein
      clearance and cholesterol redistribution. ApoE is abundantly expressed in
      astrocytes in the brain and is closely linked to the pathogenesis of
      Alzheimer's disease (AD). We report here that small molecule ligands that
      activate either liver X receptors (LXR) or retinoid X receptor (RXR) lead
      to a dramatic increase in apoE mRNA and protein expression as well as
      secretion of apoE in a human astrocytoma cell line (CCF-STTG1 cells).
      Examination of primary mouse astrocytes also revealed significant
      induction of apoE mRNA, and protein expression and secretion following
      incubation with LXR/RXR agonists. Moreover, treatment of mice with a
      specific synthetic LXR agonist T0901317 resulted in up-regulation of apoE
```

```
      mRNA and protein in both hippocampus and cerebral cortex, indicating that
      apoE expression in brain can be up-regulated by LXR agonists in vivo.
      Along with a dramatic induction of ABCA1 cholesterol transporter
      expression, these ligands effectively mediate cholesterol efflux in both
      CCF-STTG1 cells and mouse astrocytes in the presence or absence of
      apolipoprotein AI (apoAI). Our studies provide strong evidence that small
      molecule LXR/RXR agonists can effectively mediate apoE synthesis and
      secretion as well as cholesterol homeostasis in astrocytes. LXR/RXR
      agonists may have significant impact on the pathogenesis of multiple
      neurological diseases, including AD.
AD  - Lilly Research Laboratories, Eli Lilly & Company, Indianapolis, Indiana,
      USA State University of New York (SUNY) Downstate Medical Center,
      Brooklyn, New York, USA.
FAU - Liang, Yu
AU  - Liang Y
FAU - Lin, Suizhen
AU  - Lin S
FAU - Beyer, Thomas P
AU  - Beyer TP
FAU - Zhang, Youyan
AU  - Zhang Y
FAU - Wu, Xin
AU  - Wu X
FAU - Bales, Kelly R
AU  - Bales KR
FAU - DeMattos, Ronald B
AU  - DeMattos RB
FAU - May, Patrick C
AU  - May PC
FAU - Li, Shuyu Dan
AU  - Li SD
FAU - Jiang, Xian-Cheng
AU  - Jiang XC
FAU - Eacho, Patrick I
AU  - Eacho PI
FAU - Cao, Guoqing
AU  - Cao G
FAU - Paul, Steven M
AU  - Paul SM
LA  - eng
PT  - Journal Article
PL  - England
TA  - J Neurochem
JID - 2985190R
SB  - IM
EDAT- 2004/01/15 05:00
MHDA- 2004/01/15 05:00
AID - 2183 [pii]
PST - ppublish
SO  - J Neurochem 2004 Feb;88(3):623-34.
```

A.2 LocusLink

Voici un exemple d'enregistrement LocusLink (chaque champ est sur une lignes, mais nous les avons parfois coupées pour des raisons de lisibilité).

```
>>3638
LOCUSID: 3638
LOCUS_CONFIRMED: yes
LOCUS_TYPE: gene with protein product, function known or inferred
ORGANISM: Homo sapiens
```

```
STATUS: REVIEWED
NM: NM_005542|38327527|na
NP: NP_005533|28882053
CDD: KOG4363: Putative growth response protein [Signal transduction
  mechanisms]|22139|679|na|2.657410e+02
PRODUCT: insulin induced gene 1 isoform 1
TRANSVAR: Transcript Variant: This variant (1) lacks an exon within
  the coding region compared to variant 2. The resulting isoform (1) has
  a shorter and distinct C-terminus, as compared to isoform 2.
ASSEMBLY: AA565248,AF086365,AK095977,BC001880,BI550548
NM: NM_198336|38327528|na
NP: NP_938150|38327529
CDD: KOG4363: Putative growth response protein [Signal transduction
  mechanisms]|22139|674|na|2.638150e+02
PRODUCT: insulin induced gene 1 isoform 2
TRANSVAR: Transcript Variant: This variant (2) represents the longest
  transcript and encodes the longest isoform (2).
ASSEMBLY: AA565248,AF086365,AK095977,BC001880,BI550548,BQ059075
NM: NM_198337|38327530|na
NP: NP_938151|38327531
CDD: KOG4363: Putative growth response protein [Signal transduction
  mechanisms]|22139|226|na|9.124540e+01
PRODUCT: insulin induced gene 1 isoform 3
TRANSVAR: Transcript Variant: This variant (3) lacks multiple exons
  compared to variant 2, which causes a translation frameshift. The
  resulting isoform (3) has a distinct and shorter C-terminus, as
  compared to isoform 2.
ASSEMBLY: AA565248,AF086365,AK095977,BC001880,BI550548,BI918342
CONTIG: NT_034885.3|37538187|na|718914|729814|+|7|reference
EVID: supported by alignment with mRNA
XM: NM_005542|38327527|na
XP: NP_005533|28882053|na
CONTIG: NT_079596.1|37539429|na|54344506|54355411|+|7|Toronto
EVID: supported by alignment with mRNA
XM: NM_005542|38327527|na
XP: NP_005533|28882053|na
ACCNUM: U96876|2358268|na|na|na
TYPE: g
PROT: AAB69121|2358269|1
ACCNUM: AA565248|2336887|na|na|na
TYPE: m
ACCNUM: AF086365|3483710|na|59|521
TYPE: m
ACCNUM: AK095977|21755347|na|803|2261
TYPE: m
ACCNUM: AY112745|21314230|na|na|na
TYPE: m
PROT: AAM44086|21314231|1
ACCNUM: BC001880|12804864|na|na|na
TYPE: m
PROT: AAH01880|12804865|1
ACCNUM: BI550548|15437860|na|na|na
TYPE: m
ACCNUM: BI918342|16199853|na|na|na
TYPE: m
ACCNUM: BQ059075|19818415|na|na|na
TYPE: m
ACCNUM: BT007227|30583292|na|na|na
TYPE: m
PROT: AAP35891|30583293|1
ACCNUM: none|na|na|na|na
```

```
TYPE: p
PROT: O15503|20141473|0
OFFICIAL_SYMBOL: INSIG1
OFFICIAL_GENE_NAME: insulin induced gene 1
ALIAS_SYMBOL: CL-6
ALIAS_SYMBOL: MGC1405
PREFERRED_PRODUCT: insulin induced gene 1 isoform 1
PREFERRED_PRODUCT: insulin induced gene 1 isoform 2
PREFERRED_PRODUCT: insulin induced gene 1 isoform 3
SUMMARY: Summary: Oxysterols regulate cholesterol homeostasis through
  liver X receptor (LXR) and sterol regulatory element-binding protein
  (SREBP) mediated signaling pathway. This gene is an insulin-induced
  gene. It encodes an endoplasmic reticulum (ER) membrane protein that
  plays a critical role in regulating cholesterol concentrations in
  cells. This protein binds to the sterol-sensing domains of SREBP
  cleavage-activating protein (SCAP) and HMG CoA reductase, and is
  essential for the sterol-mediated trafficking of the two
  proteins. Alternatively spliced transcript variants encoding
  distinct isoforms have been observed.
CHR: 7
STS: WI-11991|7|6079|na|seq_map|epcr
STS: WI-13602|-|45347|na|na|epcr
ALIAS_PROT: INSIG-1 membrane protein
BUTTON: unigene.gif
LINK: http://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=416385
UNIGENE: Hs.416385
OMIM: 602055
MAP: 7q36|RefSeq|C|
MAPLINK: default_human_gene|INSIG1
BUTTON: snp.gif
LINK: http://www.ncbi.nlm.nih.gov/SNP/snp_ref.cgi?locusId=3638
BUTTON: homol.gif
LINK: http://www.ncbi.nlm.nih.gov/HomoloGene/homolquery.cgi?TEXT=3638[loc]&TAXID=9606
BUTTON: gdb.gif
LINK: http://www.gdb.org/gdb-bin/genera/accno?GDB:6108060
BUTTON: ensembl.gif
LINK: http://www.ensembl.org/Homo_sapiens/contigview?geneid=NM_198336
BUTTON: ucsc.gif
LINK: http://genome.ucsc.edu/cgi-bin/hgTracks?org=human&position=NM_198336
BUTTON: mgc.gif
LINK: http://mgc.nci.nih.gov/Genes/GeneInfo?ORG=Hs&CID=416385
PMID: 12963821,12869692,12535518,12477932,12242342,12202038,12115587,9268630
GRIF: 12242342|INSIG-1 plays a role in regulating cholesterol
  concentration in human cells
GRIF: 12869692|insig-1 expression restricts lipogenesis in mature
  adipocytes and blocks differentiation in preadipocytes
GRIF: 12115587|INSIG1 and p41 Arp2/3 complex (p41-Arc)reduced
  expression might be involved in gastric cancer development or
  progression
GRIF: 12535518|Data show that Insig-1 appears to play an essential
  role in the sterol-mediated trafficking of two proteins with
  sterol-sensing domains, HMG CoA reductase and SCAP.
GO: molecular function|DNA binding|IEA|GO:0003677|GOA|na
GO: biological process|regulation of transcription, DNA-dependent|IEA|GO:0006355|GOA|na
GO: biological process|metabolism|TAS|GO:0008152|GOA|9268630
GO: biological process|cell proliferation|TAS|GO:0008283|GOA|9268630
GO: cellular component|nucleus|IEA|GO:0005634|GOA|na
```

A.3 OMIM

La figure 6 donne un exemple de page OMIM.

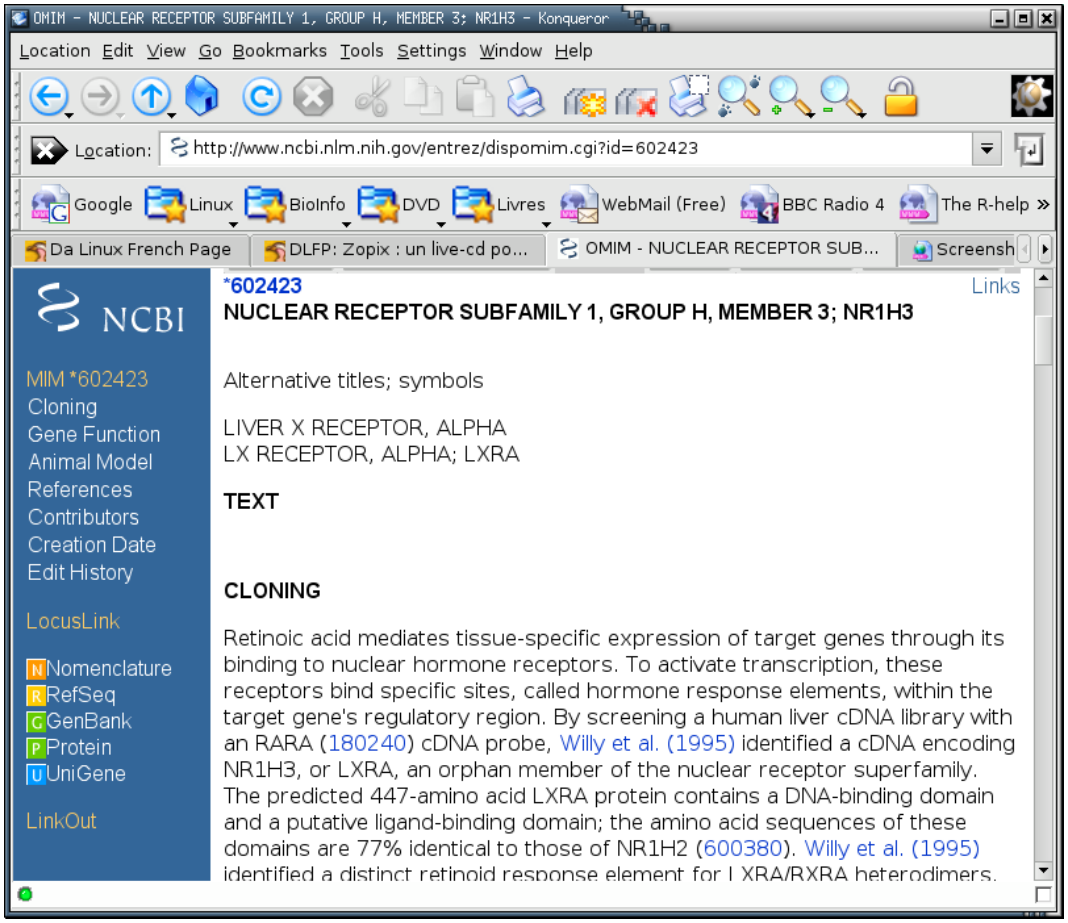

Fig. 6 – Un exemple de page OMIM

A.4 Gene Ontology

Voici un morceau de la Gene Ontology (les lignes commençant par ... sont celles que nous avons du couper).

```
!autogenerated-by:     DAG-Edit version 1.409-beta-4
!saved-by:             midori
!date:                 Fri Jan 09 14:58:14 GMT 2004
!version: $Revision: 2.443 $
!type: % is_a is a
!type: < part_of Part of
$Gene_Ontology ; GO:0003673
 <cellular_component ; GO:0005575
  %cell ; GO:0005623
   <bud ; GO:0005933
    <bud neck ; GO:0005935 % site of polarized growth (sensu Saccharomyces) ; GO:0000134

     <contractile ring (sensu Saccharomyces) ; GO:0000142 ;
       ... synonym:cytokinetic ring (sensu Saccharomyces) ; synonym:neck
       ... ring % contractile ring (sensu Fungi) ; GO:0030480
      <septin ring (sensu Saccharomyces) ; GO:0000144 % septin ring (sensu Fungi) ; GO:0030481
     <polarisome ; GO:0000133 < bud tip ; GO:0005934 < cell cortex ; GO:0005938
    <bud tip ; GO:0005934 % site of polarized growth (sensu Saccharomyces) ; GO:0000134
     <polarisome ; GO:0000133 < bud neck ; GO:0005935 < cell cortex ; GO:0005938
   <cell body ; GO:0043025
   <cell fraction ; GO:0000267
    %insoluble fraction ; GO:0005626 ; synonym:particle-bound
    %membrane fraction ; GO:0005624
     %integral to membrane of membrane fraction ; GO:0000299
     %peripheral to membrane of membrane fraction ; GO:0000300
     %synaptosome ; GO:0019717
     %vesicular fraction ; GO:0042598
      %microsome ; GO:0005792
       %rough microsome ; GO:0019718
       %smooth microsome ; GO:0019719
    %soluble fraction ; GO:0005625 ; synonym:soluble
```

B Code

B.1 R

Voici le code utilisé pour trouver, en tatonnant, le coefficient de la fonction sigmoïdale qui donne la répartition la plus uniforme de la "probabilité" déduite du coefficient IVI.

```
library(tkrplot)
animate <- function (plot.function, limits) {
  bb <- mean(limits)
  tt <- tktoplevel()
  img <-tkrplot(tt, function () { plot.function(bb) } )
  f <- function (...) {
    b <- as.numeric(tclvalue("bb"))
    if (b != bb) {
      bb <<- b
      tkrreplot(img)
    }
  }
  s <- tkscale(tt, command=f,
               from=limits[1], to=limits[2], variable="bb",
               showvalue=TRUE,
               resolution=diff(range(limits))/100,
               orient="horiz")
  tkpack(img,s)
}
```

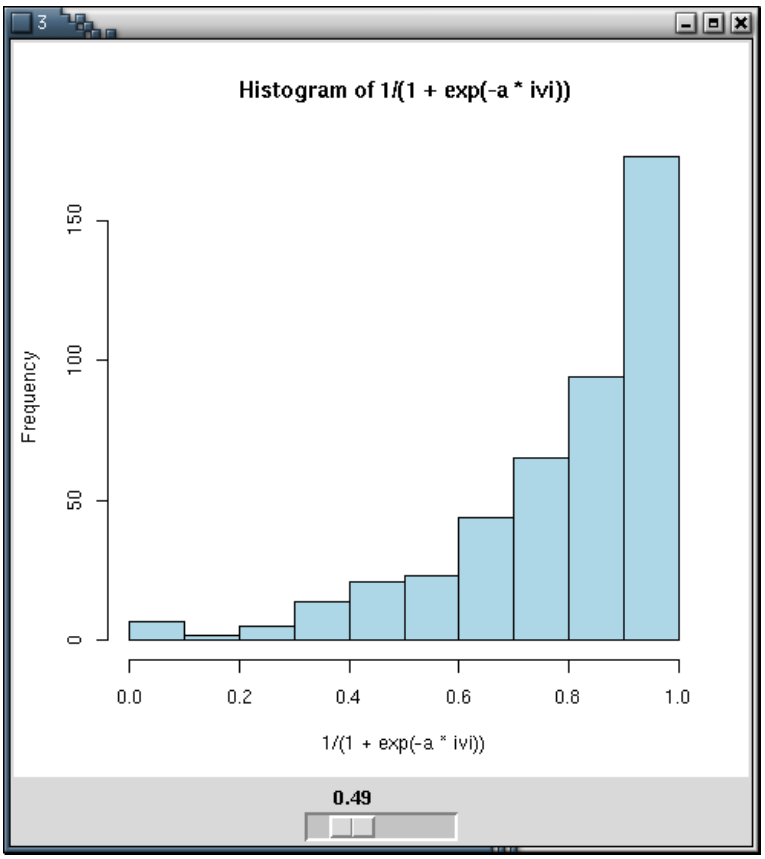

FIG. 7 – La fenêtre permettant de trouver, en tatonnant, le coefficient de la fonction sigmoïde qui donne la répartition la plus uniforme de la pseudo-probabilité obtenue à partir du score IVI.

```
animate(
  function (a) {
    hist(1/(1+exp(-a*ivi)),col='light blue')
  },
  c(.1,2)
)
```

Voici le code utilisé pour réaliser la régression non-linéaire permettant de trouver ce coefficient en comparant l'indice IVI et la probabilité donnée par le classificateur de Bayes.

```
> library(nls)
> sigmoid <- function(x, k = 1) {
+     1/(1 + exp(-k * x))
+ }
> r <- nls(Bayes ~ sigmoid(IVI, k), start = list(k = 1), data = x)
> r$m$getPars()

       k
1.215627
```

B.2 Perl

Pour réaliser diverse tâches, nous avons utilisé Perl ou des outils Unix classiques. En voici quelques exemples.

LocusLink est un fichier énorme, nous n'avons en fait utilisé qu'une petite partie des informations qu'il contient sur chaque gène. Cet élaguage de LocusLink a été réalisé ainsi.

```
gzip -dc LL_tmpl.gz |
  perl -n -e 'print if m/^(>>|OFFICIAL|ALIAS|PREFERED|OMIM|ORGANISM|PRODUCT|LOCUSID)/'
```

Les résumés nous étaient fournis dans un fichier HTML : les commandes suivantes permettent de récupérer le texte et le numéro des articles, puis de mettre tous les articles dans un répertoire, avec un article par fichier.

```
for i in `lynx -dump 2004_01_23_biomail_01.htm |
  grep Abstract |
  perl -p -e 's/^\s*[0-9]+\.\s+//'`
do
  GET $i
done > 2004_01_23_biomail_01_abstract.htm
perl -n -l -e '
  BEGIN {undef$/}
  while( s/([^>]+)<br><br>PMID:\s+([0-9]+)// ){
    print "\n$2\n$1"
  }' < bibliothesaurus[12].htm <2004_01_23_biomail_01_abstract.htm > articles.txt
perl -p -e 'BEGIN{undef$/}s/^([0-9]+)\n(.*)/echo "$2" > $1/gm' <articles.txt | sh
```

Les commandes SQL produites par notre programme forment un fichier énorme, que l'on peut découper en petits morceaux ainsi :

```
split -l 30000 <résultat.sql
```

B.3 Python

Voici maintenant le cœur du programme.

```
#! /usr/bin/python
# -*- coding: iso-8859-1 -*-

# (c) 2004 Franck CHANTELOT, Alexandre CLAUDE, Vincent ZOONEKYND
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA


##############################################################
## HISTORY
## 2004/03/07 Affichage du code R correspondant aux données brutes du calcul
##         des taux de précision et de rappel
## 2004/03/06 Calcul des taux de rappel et de précision
##         Lemmatisation des phrases pour le calcul des probabilités
##         Correction du découpage en phrases : on tient compte des initiales.
## 2004/03/03 Ajout du numéro de l'enregistrement LocusLink
## 2004/02/23 Correction de bug : "ii+1" au lieu de "ii" ("RXR" interagit avec "RXR")
## 2004/02/22 La fonction uniq préserve l'ordre
##         On tient aussi compte des phrases comportant plus de
##         deux noms de gènes
##         Rédaction des commentaires
## 2004/02/21 Classificateur de Bayes naïf (Alexandre)
##         Correction d'un bug dans le découpage en phrases
##         Comparaison de la probabilité donnée par le
##         classificateur de Bayes et l'indice IVI.
##         Choix du paramètre de la fonction sigmoïde pour
##         transformer l'indice IVI en quelque-chose qui ressemble
##         à une probabilité.
## 2004/02/17 Noms de gènes avec des espaces
##         On essaye de trouver le nom le plus long possible
##         SQL: LocusLink
##         SQL: Texte
```

```
45 ##         Les abstracts sont dans un répertoire, un abstract par fichier
46 ## 2004/02/16 Réécriture du parseur pour qu'il tienne compte d'OMIM et de l'espèce
47 ##         On vire la drosophile et ses noms de gènes bizarres
48 ##         J'enlève les noms de gènes qui sont aussi des mots courrants
49 ##         Découpage en mots : je tiens compte des parenthèses
50 ##         (mais on peut avoir des parenthèses en plein milieu d'un
51 ##         symbole : je demande que les parenthèses soient
52 ##         suivies/précédées d'un espace, sauf en fin de phrase)
53 ###############################################################
54
55 ###############################################################
56 ##
57 ## Paramètres
58 ##
59 ###############################################################
60
61 ## Nom du fichier contenant LocusLink.
62 ##    Nous utilisons ici une version "simplifiée", dans laquelle on a
63 ##    retenu que les champs qui nous intéressent.
64 ##    Elle a été obtenue à l'aide de la commande :
65 ##      gzip -dc LL_tmpl.gz |
66 ##      perl -n -e 'print if m/^(>>|OFFICIAL|ALIAS|PREFERED|OMIM|ORGANISM|PRODUCT)/'
67 ##    Le programme fonctionnerait aussi avec le fichier complet, qui
68 ##    est beaucoup plus gros (quelques centaines de Mo, au lieu de
69 ##    quelques dizaines).
70 ##    Vous pouvez récupérer le fichier original ici :
71 ##      ftp://ftp.ncbi.nih.gov/refseq/LocusLink/LL_tmpl.gz
72 locuslink = "LL_tmpl_simplified"
73
74 ## Corpus d'apprentissage
75 ##    Le corpus d'apprentissage est découpé en deux morceaux : dans un
76 ##    premier fichier, les phrases décrivant une interaction, dans un
77 ##    second, les phrases n'en décrivant pas. On se limite aux phrases
78 ##    comportant au moins deux noms de gènes.
79 interactions = "interactions.txt"
80 pas_interaction = "pas_interaction.txt"
81
82 ## Corpus
83
84 ##    Répertoire contenant les résumées des articles : un article par
85 ##    fichier, le nom du fichier est le numéro PubMed de l'article.  Il
86 ##    a été obtenu de la manière suivante, à l'aide d'un fichier
87 ##    NUMEROS, contenant les numéros PubMed des articles qui nous
88 ##    intéressent.
89 ##      for i in `cat NUMEROS`
90 ##      do
91 ##      GET $i
92 ##      done > abstracts.html
93 ##      perl -n -l -e '
94 ##      BEGIN{undef$/}
95 ##      while( s/([^>]+)<br><br>PMID:\s+([0-9]+)// ){
96 ##        print "\n$2\n$1"
97 ##      }' < abstracts.html > articles.txt
98 ##      mkdir articles/
99 ##      perl -p -e 'BEGIN{undef$/}s/^([0-9]+)\n(.+)/echo "$2" > $1/gm' <../articles.txt | sh
100 test = "articles"
101
102 ## Quelques bibliothèques
103 from nltk.stemmer.porter import PorterStemmer
104 stemmer = PorterStemmer()
105 from nltk.token import Token
106 from math import log, exp
107 import re
108 import gzip
109
110 ###############################################################
```

```
111 ##
112 ## Diverses fonctions utilitaires
113 ##
114 ################################################################
115
116 ## uniq
117 ##   Analogue de la commande Unix "uniq" : cette fonction enlève les
118 ##   doublons d'une liste. A la différence de la commande Unix, elle
119 ##   ne suppose pas que la liste est triée. Elle préserve l'ordre des
120 ##   éléments : seule la première occurrence de chacun sera préservée.
121 def uniq(a):
122   seen = {}
123   r = []
124   for x in a:
125     if not seen.has_key(x):
126       seen[x] = 1
127       r.append(x)
128   return r
129
130 ## text_split
131 ##   Fonction pour découper une phrase en mots.
132 ##   La manière la plus simple de procéder semble être de couper au
133 ##   niveau des espaces -- mais cela pose des problèmes, à cause des
134 ##   caractères de ponctuation ou des parenthèses. L'expression
135 ##   régulière suivante en tient compte.
136 ##   Remarque : Lors de ce découpage, nous n'avons pas tenu compte des
137 ##   slashes (/), car il apparaissent parfois à l'intérieur d'un
138 ##   gène. Mais cela pose des problèmes, car ils sont parfois utilisés
139 ##   pour séparer des noms de gènes.
140 def text_split(a):
141   return re.compile(r"\s+[,;:!?.]\s+|[,;:!?.)]\s+|\s+[(,;:!?.]|[,;:!?.)]$|\s+").split(a)
142
143 ## stop_words
144 ##   Liste de noms de gènes qui sont aussi des noms communs.
145 ##   Certains noms de gènes sont aussi des noms communs, ce qui
146 ##   perturbe grandement la reconnaissance des noms de gènes, des
147 ##   phrases comportant au moins deux noms de gènes et, finalement, la
148 ##   reconnaissance des phrases décrivant des interactions.  Nous
149 ##   choisissons de ne pas les reconnaître comme des noms de gènes.
150 stop_words = {}
151 for i in [ "accord", "ago", "band", "bare", "bran", "branch", "broad",
152   "dock", "hairless", "lag", "lethal", "light", "net", "se",
153   "smaller", "spin", "spot", "tube", "who", "zipper", "box", "disc",
154   "fish", "mg", "slight", "span", "weak", "blunt", "far",
155   "gamma1", "how", "i", "min", "out", "smooth", "us", "brown", "ear",
156   "rose", "side", "term", "thick", "up", "black", "contact", "divers",
157   "mix", "per", "repressor", "kidney", "period", "white",
158   "blot", "map", "upstream", "gel", "do", "knockout", "blood", "fat",
159   "fast", "lack", "small", "put", "brain", "act", "similar",
160   "inhibitor", "key", "did", "can", "high", "at", "be",
161   "or", "we", "not", "an", "as", "cell", "liver", "for", "receptor",
162   "with", "gene", "by", "a", "to", "in", "has", "little",
163   "X", "alpha", "beta", "gamma",
164   'a', 'and', 'in', 'of','the'
165         ]:
166   stop_words[i] = 1
167
168 ## stop_words_start
169 ##   Noms de gènes avec une majuscule qui sont aussi des noms communs
170 ##   souvent utilisés en début de phrase.
171 stop_words_start = {}
172 for i in ["The", "In", "It", "As"]:
173   stop_words_start[i] = 1
174
175 ################################################################
176 ##
```

```
177 ## Classes
178 ##
179 ################################################################
180
181 class Phrase:
182   ## Constructeur
183   ##   chaine: chaine de caractères contenant une phrase
184   ##   genes: instance de LocusLink, qui sera utilisée pour
185   ##     regroupper les noms de gènes qui comportent plusieurs mots.
186   def __init__(self,chaine, genes):
187     self._chaine = chaine.replace("\n"," ")
188     self._mots = text_split(chaine)
189     if genes:
190       self._mots = genes.preprocess_text(self._mots)
191     # Lemmatisation (algorithme de Porter, implémenté dans NLTK)
192     self._lemmes = [ stemmer.stem(Token(token)).type() for token in self._mots ]
193   ## mots()
194   ##   Retourne la liste des mots de la phrase
195   def mots(self):
196     return self._mots
197   ## lemmes()
198   ##   Retourne la liste des lemmes (i.e., des mots déconjugués) de
199   ##   la phrase
200   def lemmes(self):
201     return self._lemmes
202   ## texte()
203   ##   Retourne le texte complet de la phrase, tel qu'il a été donné
204   ##   au constructeur
205   def texte(self):
206     return self._chaine
207
208 ################################################################
209
210 class Texte:
211   ## Constructeur
212   ##   t: chaine de caractère contenant le texte à découper en phrases
213   ##   genes: instance de LocusLink qui sera utilisée pour
214   ##     reconnaître les noms de gènes en plusieurs mots dans les
215   ##     phrases.
216   def __init__(self,t,genes):
217     #self._phrases = [ Phrase(p,genes) for p in re.compile("\.\s+").split(t) ]
218     self._phrases = [ Phrase(p,genes) for p in re.compile("(?<=[^A-Z])\.\s+").split(t) ]
219     print "DEBUG (texte): Creating object with %d phrases" % len(self._phrases)
220   ## phrases()
221   ##   Retourne la liste des Phrases du texte
222   def phrases(self):
223     return self._phrases
224
225 ################################################################
226
227 ## Classificateur
228 ##   Classe abstraite décrivant l'interface des classificateurs (i.e.,
229 ##   des morceaux de programmes qui vont effectuer le tri dans les
230 ##   phrases, en repérant celles qui décrivent des interactions).
231 ##   On peut imaginer plusieurs implémentations, avec un
232 ##   classificateur de Bayes, l'indice IVI, des arbres de prédiction,
233 ##   des forêts d'arbres de prédiction (bagging, boosting), des SVM,
234 ##   etc.
235 class Classificateur:
236   ## Constructeur
237   ##   oui: Texte (corpus d'apprentissage) contenant des phrases
238   ##     décrivant des interactions
239   ##   non: texte contenant des phrases ne décrivant pas
240   ##     d'interaction
241   def __init__(self,oui,non):
242     raise "Classe abstraite"
```

```
243   ## indice()
244   ##    phrase: une Phrase
245   ##    Retourne la probabilité (ou un nombre entre 0 et 1 qu'on pourra
246   ##      interpréter comme une probabilite) que la phrase décriven
247   ##      une interaction.
248   def indice(self,phrase):
249     raise "méthode non implémentée"
250
251 ##############################################################
252
253 # Calcul de l'IVI (Indice de Vraissemblance d'Interaction), d'après
254 ##   V. Pillet, Méthodologie d'extraction automatique d'information à
255 ##   partir de la littérature scientifique en vue d'alimenter un
256 ##   nouveau système d'information, application à la génétique
257 ##   moléculaire pour l'extraction d'information sur les informations,
258 ##   thèse de doctorat, 2000
259 ##   http://ms161u13.u-3mrs.fr/memoires/ViolainePillet_T.pdf
260 class IVI(Classificateur):
261   def __init__(self,oui,non):
262     interaction = {}
263     pas_interaction = {}
264     # On compte :
265     # interaction[mot] = nombre d'occurrences de "mot" dans des
266     #   phrases décrivant des interactions
267     # pas_interaction[mot] = nombre d'occurrences de "mot" dans
268     #   des phrases ne décrivant pas d'interaction
269     for phrase in oui.phrases():
270       for mot in phrase.lemmes():
271         if not interaction.has_key(mot):
272           interaction[mot] = 0.0
273         if not pas_interaction.has_key(mot):
274           pas_interaction[mot] = 0.0
275         interaction[mot] = interaction[mot] + 1
276     for phrase in non.phrases():
277       for mot in phrase.lemmes():
278         if not interaction.has_key(mot):
279           interaction[mot] = 0.0
280         if not pas_interaction.has_key(mot):
281           pas_interaction[mot] = 0.0
282         pas_interaction[mot] = pas_interaction[mot] + 1
283     # On calcule l'indice IVI de chaque mot
284     self.ivi = {}
285     for mot in interaction.keys():
286       self.ivi[mot] = (interaction[mot] - pas_interaction[mot]) / (interaction[mot] + pas_interact
287   ## dump_words():
288   ##    Affiche l'indice IVI de chaque mot
289   ##    (uniquement à des fins de débugguage)
290   def dump_words(self):
291     for mot in self.ivi.keys():
292       print mot, ivi[mot]
293   ## indice()
294   ##    L'indice IVI d'une phrase est la somme des indices des mots
295   ##    qui la compose.
296   ##    On ne retourne pas l'indice "brut", mais on le transforme,
297   ##    par une fonction sigmoïdale, en un nombre dans [0,1], que
298   ##    l'on pourra interpréter comme une probabilité.
299   ##    Le paramètre (k=1/3) de la fonction sigmoïde a été choisi de
300   ##    sorte que les valeurs soient "bien réparties" entre 0 et 1.
301   ##    Une autre manière de le choisir aurait été de comparer
302   ##    l'indice IVI avec la probabilité donnée par le classificateur
303   ##    de Bayes : on obtiendrait k=1.2, mais les probabilités
304   ##    seraient concentrées autour de 0 et de 1.
305   def indice_raw(self,phrase):
306     i = 0.0
307     for mot in phrase.lemmes():
308       if self.ivi.has_key(mot):
```

```
309         i = i+self.ivi[mot]
310     print "IVI:", i
311     return i
312   def indice(self,phrase):
313     i = self.indice_raw(phrase)
314     return 1/(1+exp(-i/3.0))
315
316 ############################################################
317
318 ## Classificateur Bayesien naïf
319 class BAYES(Classificateur):
320   def __init__(self,oui,non):
321     phrases_int=oui.phrases()
322     nbPhraseInt = phrases_int.__len__()
323
324     phrases_no_int=non.phrases()
325     nbPhraseNoInt = phrases_no_int.__len__()
326
327     nbPhrase = nbPhraseInt + nbPhraseNoInt
328     pourcentPhraseInt = nbPhraseInt* 1. / nbPhrase
329     pourcentPhraseNoInt = nbPhraseNoInt * 1. / nbPhrase
330
331     # incrémentation de l'indice d'interaction et de non
332     # interaction dans les tableaux de cle (mot)
333     interaction = {}
334     no_interaction = {}
335
336     mots_vides = {}
337     for i in ["a", "the", "and", "an", "or", 'in', 'of']:
338       mots_vides[i]=1
339
340     for phrase in phrases_int:
341       for mot in uniq(phrase.lemmes()):
342         if not mots_vides.has_key(mot):
343           if interaction.has_key(mot):
344             interaction[mot]=interaction[mot]+1.00000/nbPhraseInt
345           else:
346             interaction[mot]=1.00000/nbPhraseInt
347
348     for phrase in phrases_no_int:
349       for mot in uniq(phrase.lemmes()):
350         if not mots_vides.has_key(mot):
351           if no_interaction.has_key(mot):
352             no_interaction[mot]=no_interaction[mot]+1.00000/nbPhraseNoInt
353           else:
354             #print no_interaction[mot], mot
355             no_interaction[mot]=1.00000/nbPhraseNoInt
356
357     #Compléter les deux tables de hachage : elles doivent avoir les meme clefs.
358     # Pour les clefs qui manquent, on mettra une valeur faible
359     # par exemple, 0.5/nombe_de_phrases
360     for mot in interaction:
361       if not mots_vides.has_key(mot):
362         if not(no_interaction.has_key(mot)):
363           no_interaction[mot] = 0.5 / nbPhrase
364
365     for mot in no_interaction:
366       if not mots_vides.has_key(mot):
367         if not(interaction.has_key(mot)):
368           interaction[mot] = 0.5 / nbPhrase
369
370     for mot in interaction:
371       print "DEBUG: Bayes", mot, "==>", interaction[mot], ",", no_interaction[mot]
372
373     self.interaction_dict = interaction
374     self.no_interaction_dict = no_interaction
```

```
375     self.pourcentPhraseInt_dict = pourcentPhraseInt
376     self.pourcentPhraseNoInt_dict = pourcentPhraseNoInt
377
378   def indice(self,phrase):
379     # Calcul de l'indice de la phrase, utilisation du pourcenage
380     # de tous les mots du corpus d'apprentissage tenir compte du
381     # dictionnaire avec pas d interaction
382     i=0 # log(probabilité que la phrase décrive une interaction (à un facteur près)
383     j=0 # log(probabilité que la phrase ne décrive pas une interaction (à un facteur près)
384     for mot in self.interaction_dict:
385       if mot in phrase.lemmes() : # qu il y ait un mot dans la liste ; .... le mot est dans la ph
386         #print "B (DEBUG) (present):", mot, "i:", self.interaction_dict[mot], "j:", self.no_inter
387         i=i+log(self.interaction_dict[mot])   # P(act | interaction)
388         j=j+log(self.no_interaction_dict[mot])   # P(act | pas interaction)
389       else:
390         #print interaction[mot], mot
391         #print "B (DEBUG) (absent):", mot, "i:", self.interaction_dict[mot], "j:", self.no_intera
392         i=i+log(1-self.interaction_dict[mot])  # P(pas "act" | interaction)
393         j=j+log(1-self.no_interaction_dict[mot])  # P(pas "act" | pas interaction)
394     i = i + log(self.pourcentPhraseInt_dict)
395     j = j + log(self.pourcentPhraseNoInt_dict)
396     resultPhrase = exp(i)/(exp(i)+exp(j))
397     print "Bayes:", resultPhrase,phrase.texte()
398     return resultPhrase
399
400 ###############################################################
401
402 class LocusLinkGenes:
403   ## self.genes : table de hachage, les clefs sont les noms de
404   ##   gènes, les valeurs, des listes de noms officiels.
405   ## self.omim : table de hachage, les clefs sont les noms officiels
406   ##   de gènes, les valeurs, les numéros OMIM correspondants.
407   ## self.organism : table de hachage, les clefs sont les noms
408   ##   officiels de gènes, les valeurs, la liste des organismes chez
409   ##   lesquels on trouve un gène avec ce nom.
410   ## self.long : table de hachage, utilisée pour trouver les gènes
411   ##   dont le nom comporte plusieurs mots ; les clefs sont des
412   ##   listes de listes. Exemple :
413   ##   'insulin' => [['insulin', 'induced', 'gene', '1'],
414   ##           ['insulin', 'induced', 'gene', '1', 'isoform', '3'],
415   ##           ['insulin', 'receptor', 'substrate', '1'],
416   ##           ['insulin', 'QTL', '7'],
417   ##           ['insulin', 'receptor', 'tyrosine', 'kinase', 'substrate', 'homolog'],
418   ##           ...
419   ##           ]
420   ## Constructeur
421   ##   fichier: fichier LocusLink, éventuellement simplifié, comme
422   ##     détaillé plus haut.
423   def __init__(self,fichier):
424     f = open(fichier, 'r')
425     self.genes = {}
426     self.omim = {}
427     self.organism = {}
428     alias = re.compile(r"((OFFICIAL|PREFERRED|ALIAS)_(SYMBOL|GENE_NAME|PROT|PRODUCT)|ORGANISM|PROD
429     courrant = False
430     valeurs = []
431     organism = False
432     omim = ""
433     locusid = ""
434     nom = False
435     s = f.readline()
436     while s:
437       if s.find(">>") == 0:
438         if courrant and organism != 'Drosophila melanogaster':
439           #print "Gène :", courrant
440           #print "  Synonymes :", "; ".join(valeurs)
```

```
441           #print "  Organisme :", organism
442           #print "  OMIM :", omim
443           if not nom:
444             nom = courrant
445           nom = nom + " (" + organism + ")"
446           print "SQL:", "INSERT INTO element (name,link,symbol,locuslink) VALUES",
447           print '("' + nom + '", "' + omim + '", "',
448           print courrant + '", "' + locusid +'");'
449           for i in valeurs:
450             if not stop_words.has_key(i):
451               if not self.genes.has_key(i):
452                 self.genes[i] = []
453               self.genes[i].append(courrant)
454             print "SQL: INSERT INTO synonym (name, name_used) VALUES",
455             print '("' + nom + '", "' + i + '");'
456           self.omim[courrant] = omim
457           if self.organism.has_key(courrant):
458             print "WARNING: duplicated official name '%s' for species %s and %s" % (courrant, self
459           else:
460             self.organism[courrant] = []
461           self.organism[courrant].append(organism)
462         courrant = False
463         valeurs = []
464         organism = False
465         omim = ""
466         locusid = ""
467         nom = False
468       a = alias.search(s)
469       if a and a.group(4) != '':
470         #print "*line* ", s,
471         #print "DEBUG 1", a.group(1)
472         #print "DEBUG 2", a.group(2)
473         #print "DEBUG 3", a.group(3)
474         #print "DEBUG 4", a.group(4)
475         if ( a.group(2) == "OFFICIAL" or a.group(2) == "PREFERRED" ) and not courrant:
476           courrant = a.group(4)
477           valeurs.append(courrant)
478         elif ( a.group(2) == "OFFICIAL" or a.group(2) == "PREFERRED" ) and not nom:
479           nom = a.group(4)
480           valeurs.append(nom)
481         elif a.group(1) == "ORGANISM":
482           organism = a.group(4)
483         elif a.group(1) == "OMIM":
484           omim = a.group(4)
485         elif a.group(1) == "LOCUSID":
486           locusid = a.group(4)
487         elif not re.compile(r"^[0-9]+$").match(a.group(4)):
488           #print "  DEBUG: appending", a.group(4)
489           valeurs.append(a.group(4))
490       s = f.readline()
491     # On s'intéresse aux noms de gènes qui font plusieurs mots...
492     print "(LocusLink) Preprocessing long gene names"
493     self.long = {}
494     for nom in self.genes.keys():
495       mots = nom.split()
496       if len(mots) > 1:
497         #print "DEBUG: (LocusLink, long gene names)", nom, "==>", mots
498         if not self.long.has_key(mots[0]):
499           self.long[mots[0]] = []
500         self.long[mots[0]].append(mots)
501     # On les met dans l'ordre : les plus longs en premier
502     print "(LocusLink) Preprocessing long gene names (2)"
503     for nom in self.long.keys():
504       #print "  DEBUG (LocusLink, long gene names, 2)", nom, "was", self.long[nom]
505       # En Perl, on pourrait écrire ça en une seule ligne...
506       tmp = [ (len(a),a) for a in self.long[nom] ] # Transformation de Schwartz
```

```
507         tmp.sort()
508         tmp.reverse()
509         self.long[nom] = [ y for (x,y) in tmp ]
510         print "  DEBUG (LocusLink, long gene names, 2)", nom, "==>", self.long[nom]
511     ## preprocess_text()
512     ##   phrase: une liste de mots (pas encore une Phrase)
513     ##   On tente de regroupper ces mots afin de former des noms de
514     ##   gènes, car certains noms de gènes sont composés de plusieurs
515     ##   mots.
516     ##   Dans une version ultérieure du programme, on pourrait mettre
517     ##   cette fonction dans une autre classe, car elle pourrait
518     ##   s'appliquer à des locutions (de novo, a priori, etc.), ou à
519     ##   des collocations.
520     def preprocess_text(self,phrase):
521       #print "DEBUG: Preprocessing Text", " ".join(phrase)
522       n = len(phrase)
523       res = []
524       i = 0
525       while i<n:
526         if self.long.has_key(phrase[i]):
527           #print "  DEBUG: (Preprocessing Text) FOUND:", phrase[i]
528           for s in self.long[phrase[i]]:
529             ok = True
530             j = 0
531             while ok and j<len(s) and i+j<n:
532               if s[j] != phrase[i+j]:
533                 ok = False
534               j = j+1
535             ok = ok and j==len(s)
536             if ok:
537               print "DEBUG: (Preprocessing text) JOINING:", " ".join(phrase[i:(i+len(s))])
538               res.append(" ".join(phrase[i:(i+len(s))]))
539               i = i + len(s)
540               break
541           if ok:
542             break
543         res.append(phrase[i])
544         i = i + 1
545       return res
546     def termes(self,phrase):
547       """Donne la liste des termes présents dans la phrase et correspondant à des gènes"""
548       r = []
549       n = 0
550       for i in phrase.mots():
551         if (n>0 or not stop_words_start.has_key(i)) and self.genes.has_key(i):
552           r.append(i)
553         n = n + 1
554       return uniq(r)
555       #return uniq([ mot for mot in phrase.mots() if self.genes.has_key(mot) ])
556     def compte(self,phrase):
557       """Donne le nombre de gènes mentionnés dans la phrase"""
558       return len(self.termes(phrase))
559     def cherche(self,phrase):
560       """Renvoie une table de hachage dont les clefs sont les termes
561       rencontrés dans la phrase et les valeurs des listes contenant
562       les noms officiels"""
563       r = {}
564       for i in self.termes(phrase):
565         r[i] = self.genes[i]
566       return r
567     def cherche_old(self,phrase):
568       return [ self.genes(mot) for mot in
569            [ mot for mot in phrase.mots() if self.genes.has_key(mot) ] ]
570
571 ###############################################################
572
```

```
573 ## Corps du programme
574
575 print "Lecture des noms de gènes (long)"
576 genes = LocusLinkGenes(locuslink)
577
578 print "Lecture du corpus d'apprentissage"
579 i = open(interactions, "r")
580 oui = Texte(i.read(),genes)
581 i.close()
582 j = open(pas_interaction, "r")
583 non = Texte(j.read(),genes)
584 j.close()
585
586 print "DEBUG (Corpus d'apprentissage): %d phrases décrivant des interactions" % len(oui.phrases())
587 print "DEBUG (Corpus d'apprentissage): %d phrases n'en décrivant pas " % len(non.phrases())
588
589 print "Apprentissage (IVI)"
590 ivi = IVI(oui,non)
591 print "Apprentissage (Classificateur de Bayes)"
592 bayes = BAYES(oui,non)
593
594 ############################################################
595
596 print "Calcul des taux de précision et de rappel"
597
598 i1 = open("interactions_1.txt", "r")
599 o1 = Texte(i1.read(),genes)
600 i1.close()
601
602 i2 = open("interactions_2.txt", "r")
603 o2 = Texte(i2.read(),genes)
604 i2.close()
605
606 j1 = open("pas_interaction_1.txt", "r")
607 n1 = Texte(j1.read(),genes)
608 j1.close()
609
610 j2 = open("pas_interaction_2.txt", "r")
611 n2 = Texte(j2.read(),genes)
612 j2.close()
613
614 i = IVI(o1,n1)
615 b = BAYES(o1,n1)
616
617 boo=bon=bno=bnn= ioo=ion=ino=inn= 0.0
618 print "R: d <- matrix(c("
619 for p in o2.phrases():
620   print "R: %e, %e, 1," % ( i.indice_raw(p), b.indice(p) )
621   if i.indice(p) > .5:
622     ioo=ioo+1
623   else:
624     ion=ion+1
625   if b.indice(p) > .5:
626     boo=boo+1
627   else:
628     bon=bon+1
629 for p in n2.phrases():
630   print "R: %e, %e, 0," % ( i.indice_raw(p), b.indice(p) )
631   if i.indice(p) > .5:
632     isno=ino+1
633   else:
634     inn=inn+1
635   if b.indice(p) > .5:
636     bno=bno+1
637   else:
638     bnn=bnn+1
```

```
639 print "R: ),"
640 print "R: nc=3, byrow=T)"
641 print "R: colnames(d) <- c('ivi', 'bayes', 'interaction')"
642 print "R: d <- as.data.frame(d)"
643 print "R: d$interaction <- factor(ifelse(d$interaction==1,'oui','non'))"
644 print "R: plot( bayes ~ ivi, data=d, col=c('red','blue')[interaction] )"
645 print "R: library(nls)"
646 print "R: sigmoid <- function (x,k=1) { 1/(1+exp(-k*x)) }"
647 print "R: nls( bayes ~ sigmoid(ivi,k), start=list(k=1), data=d )"
648 print "R: plot( bayes ~ sigmoid(ivi,1.3), data=d, col=c('red','blue')[interaction] )"
649 print "Bayes:  Précision=%d, Rappel=%d" % ( boo/(boo+bon), boo/(boo+bno) )
650 print "Bayes:  oo=%d, on=%d, no=%d, nn=%d" % (boo,bon,bno,bnn)
651 print "IVI:    Précision=%d, Rappel=%d" % ( ioo/(ioo+ion), ioo/(ioo+ino) )
652 print "IVI:    oo=%d, on=%d, no=%d, nn=%d" % (ioo,ion,ino,inn)
653
654 ##############################################################
655
656 print "Lecture du corpus de test"
657 import os
658 for ref in os.listdir(test):
659   print "Reading article", ref
660   i = open(test + "/" + ref)
661   t = Texte(i.read(),genes)
662   i.close()
663   for p in t.phrases():
664     print
665     print p.texte()
666     print genes.compte(p), genes.termes(p)
667     print genes.cherche(p)
668     n = genes.compte(p)
669     if n >1:
670       print "PHRASE:", ref
671       print "PHRASE:", genes.termes(p)
672       print "PHRASES:", p.texte()
673       print "PHRASE: "
674       m = genes.termes(p)
675       i = ivi.indice(p)
676       b = bayes.indice(p)
677       for ii in range(n):
678         for jj in range(ii+1,n):
679           print "DEBUG: ii=%d, jj=%d, m[ii]='%s', m[jj]='%s'"%(ii,jj,m[ii],m[jj])
680           print "SQL: INSERT INTO score (num_interaction, element_1, element_2, score_ivi, score_b
681           print '(0, "' + m[ii] + '", "' + m[jj] + '", "' + str(i) + '", "' + str(b) + '", "' + p.
```

Sommaire