# Travaux Pratiques de Bioinformatique

## L'Intelligence Artificielle au service de la Génomique

Nicolas Parisot & Sergio Peignier

(D'après un notebook de Thomas Nelson)

La biologie, et en particulier les sciences omiques (*i.e.* génomique, transcriptomique, métagénomique, etc.), génèrent une très grande quantité de données (*big data*) hétérogènes qui visent à expliquer des processus cellulaires et moléculaires complexes. Face à cette avalanche de données, les outils d'apprentissage automatique (*machine learning*) s'avèrent être de précieux alliés pour extraire de l'information : la prédiction des sites de liaison des protéines, la classification des types de cancer, la prédiction de l'évolution des maladies, la médecine de précision, les études d'association à l'échelle du génome, la classification des images, le débruitage et le traitement des images au microscope etc.

L'objectif de ce TP est de se familiariser avec les concepts de *machine learning*, et notamment savoir évaluer la qualité d'un modèle obtenu par *machine learning*. Lors de ce TP, nous appliquerons le *machine learning* à l'annotation fonctionnelle des séquences génomiques. Nous implémenterons et évaluerons la qualité d'un modèle de prédiction de la fonction biologique des gènes sur la base de leur séquence nucléotidique.

Pour ce TP, vous aurez à construire un Jupyter Notebook en Python à partir des instructions ci-dessous.

### 1. Données disponibles

Vous avez à votre disposition un fichier `human_data.fasta` qui contient 4380 séquences nucléotidiques de gènes humains. La fonction biologique de chacun de ces gènes est connue et elle est mentionnée dans la description de la séquence sous la forme d'un chiffre entre 0 et 6 :

| Gene family | Number | Class label |
|---|---|---|
| G protein coupled receptors | 531 | 0 |
| Tyrosine kinase | 534 | 1 |
| Tyrosine phosphatase | 349 | 2 |
| Synthetase | 672 | 3 |
| Synthase | 711 | 4 |
| Ion channel | 240 | 5 |
| Transcription factor | 1343 | 6 |

**La première étape de votre notebook consiste donc à importer ces informations dans deux listes ordonnées de la même façon :**

| **list_sequences** | | | **list_class** | |
|---|---|---|---|---|
| index | [string] | | index | [char/int] |
| 0 | séquence nucléotidique<br>de la 1ère séquence du fichier | | 0 | classe<br>de la 1ère séquence du fichier |
| 1 | séquence2 | | 1 | classe2 |
| 2 | séquence3 | | 2 | classe3 |
| ... | ... | | ... | ... |

## 2. Recherche d'information

Dans notre étude, l'hypothèse sous-jacente à l'utilisation d'un modèle de classification repose sur le fait que la séquence nucléotidique d'un gène détermine la fonction biologique de ce gène. Il nous faut donc une méthode pour extraire de l'information à partir des séquences nucléotidiques.

L'analyse de texte est un domaine d'application majeur du *machine learning* et de nombreuses méthodes pour extraire de l'information numérique à partir de texte ont déjà été développées. La méthode la plus intuitive pour transformer un texte en information numérique repose sur le comptage du nombre d'occurrences de chaque mot présent dans le texte. Cependant, les algorithmes classiques de *machine learning* ne travaillent souvent qu'à partir de listes de tailles identiques ce qui rend leur utilisation directe impossible à partir de textes contenant un nombre de mots uniques différents. La solution est alors d'utiliser des listes contenant l'ensemble des mots du dictionnaire pour réaliser le comptage. Ainsi, même à partir de textes de tailles différentes, les listes de comptages de mots auront une taille uniforme.

Cependant, la nature de l'information textuelle que nous avons à analyser dans ce projet (i.e. séquences nucléotidiques) implique une transformation préalable.

**Implémentez une fonction permettant de renvoyer une liste de mots à partir d'une chaîne de caractères.**

A ce stade, nous sommes donc en mesure de décomposer une séquence nucléotidique en une suite de mots que nous pouvons compter pour en extraire une information numérique.

**En vous inspirant de ce qui peut être fait lors de l'analyse de textes plus conventionnels, quelle(s) proposition(s) pouvez-vous faire pour maximiser l'information extraite de nos séquences nucléotidiques étudiées ?**

Aujourd'hui, Scikit-learn est la bibliothèque de référence pour le *machine learning* en Python. Scikit-learn propose un panel très large de méthodes pour la transformation des données initiales, l'apprentissage supervisé et nonsupervisé, la sélection et la validation des modèles ou encore la visualisation des données.

**Utilisez la bibliothèque Scikit-learn, et en particulier la fonction CountVectorizer, pour implémenter la solution retenue.**

## 3. Construction du modèle de *machine learning*

Nous savons donc maintenant comment transformer nos 4380 séquences humaines en une information numérique qui peut alors être utilisée pour construire un modèle de machine learning afin de prédire la fonction biologique des gènes sur la base de leur séquence nucléotidique. Le modèle utilisé est un classifieur bayésien naïf. Il s'agit d'une méthode simple d'apprentissage supervisée basée sur le théorème de Bayes.

```
from sklearn.naive_bayes import MultinomialNB

classifier = MultinomialNB(alpha=0.1)
```

Une fois le modèle construit, il est maintenant nécessaire de l'entraîner. Nous allons donc séparer notre jeu de données en un ensemble d'entraînement et un ensemble de validation qui nous permettra d'évaluer les performances de notre modèle. Scikit-learn nous fournit des outils pour réaliser cette étape.

**Utilisez la fonction train_test_split() de Scikit-learn pour décomposer le jeu de données en 80% pour l'entraînement, 20% pour la validation.** Vous aurez besoin d'utiliser les deux objets `X` et `list_class` comme arguments ainsi que les options `test_size` pour fixer la taille du jeu d'entraînement et `random_state` pour permettre la reproductibilité de votre méthode.

Une fois le jeu d'entraînement défini, il est alors possible d'entraîner le modèle :

```
classifier.fit(X_train, y_train)
```

## 4. Evaluation des performances du modèle

Notre modèle de *machine learning* a donc été confronté à 80% du jeu de données afin d'apprendre les caractéristiques lui permettant de prédire la fonction des gènes. Nous allons maintenant appliquer le modèle sur les 20% restants du jeu de données pour évaluer ses performances.

```
y_pred = classifier.predict(X_test)
```

**A l'aide des fonctions prédéfinies de scikit-learn, évaluez les performances du modèle.**

L'accuracy est une mesure qui est classiquement utilisée pour évaluer la qualité d'un modèle. Cette mesure correspond simplement à la proportion d'objets correctement classifiés par le prédicteur. Cependant cette mesure n'est pas adaptée à certains jeux de données qui ont une distribution inégale d'objets dans chaque classe. En effet, dans un jeu de données déséquilibré ayant 1000 objets dans la classe 0 et seulement 10 objets dans la classe 1, un classifieur qui prédirait naïvement la classe 0 pour tous les objets, aurait une accuracy de 99%. Dans ce cas, il est nécessaire de considérer d'autres mesures d'évaluations, qui peuvent être définies plus clairement grâce à une matrice de confusion. Plus formellement, soit $M$ une matrice de taille $n \times n$ où $n$ est égal au nombre de classes dans le jeu de données, et $M_{i,j} \in N$ contient le nombre d'objets appartenant à la classe $i$ qui ont été classifiés comme appartenant à la classe $j$. Cette matrice nous permet de mesurer la performance de notre modèle en évaluant le nombre de classifications correctes (Vrais Positifs et Vrais Négatifs) mais également le nombre de classifications incorrectes (Faux Positifs et Faux Négatifs). Ainsi pour une classe $i \in \{1,\dots,n\}$, le nombre de vrais positifs est égal à $TP_i = N_{i,i}$ , le nombre de faux négatifs est égal à $FN_i = \sum_{j \notin i} N_{i,j}$ , le nombre de faux positifs est égal à $FP_i = \sum_{j \notin i} N_{j,i}$ , et le nombre de vrais négatifs est égal à $TN_i = \sum_{j \notin i} \sum_{k \notin i} N_{k,j}$ .
Plusieurs mesures de performance standard peuvent être calculées pour chaque classe:

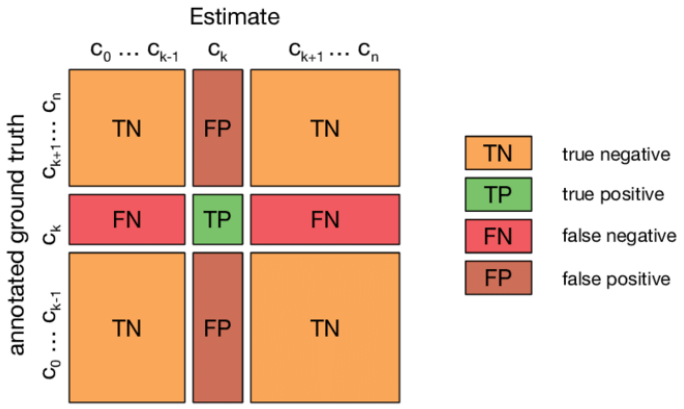
- $Rappel_i = TP_i / (TP_i + FN_i)$
- $Precision_i = TP_i / (TP_i + FP_i)$
- $F1_i = 2 \times Rappel_i \times Precision_i / (Rappel_i + Precision_i)$

Finalement il est possible de calculer des scores globaux à partir des mesures calculées pour chaque classe, suivant les trois principes suivants:

- **Micro**: Les métriques globales sont calculées en comptant le nombre total de vrais positifs, vrais négatifs et faux positifs:
    - $Rappel = \sum_i TP_i / (\sum_i TP_i + \sum_i FN_i)$
    - $Precision = \sum_i TP_i / (\sum_i TP_i + \sum_i FP_i)$
    - $F1 = 2 \times \sum_i Rappel_i \times \sum_i Precision_i / (\sum_i (Rappel_i + Precision_i))$

- **Macro** Les métriques globales sont calculées comme une moyenne non-pondérée des scores des différentes classes:
    - $Rappel \ = 1/n \times \sum_i (TP_i / (TP_i + FN_i))$
    - $Precision \ = 1/n \times \sum_i (TP_i / (TP_i + FP_i))$
    - $F1 \ = \ 2 \times Rappel \ \times Precision \ / (Rappel \ + Precision \ )$
- **Weighted:** Les métriques globales sont calculées comme une moyenne des scores des différentes classes, pondérées par le vrai nombre d'objets dans chacune des classes.

L'image suivante représente de manière schématique une table de confusion, ainsi que le nombre de classifications correctes (Vrais Positifs et Vrais Négatifs) mais également le nombre de classifications incorrectes (Faux Positifs et Faux Négatifs), associées à la classe $c_k$.




5. Application du modèle à d'autres espèces

Vous avez à votre disposition deux autres fichiers de séquences nucléotidiques issus de deux autres espèces afin d'évaluer les performances de notre modèle construit chez l'homme sur d'autres jeux de données.

Le fichier `chimp_data.fasta` contient 1682 séquences nucléotidiques issues du chimpanzé alors que le fichier `dog_data.fasta` contient 820 séquences nucléotidiques issues du chien.

**Appliquez le modèle précédemment construit sur l'Homme à ces deux jeux de données (sans le ré-entraîner !) et discutez les performances observées.**