



PROJET INDIVIDUEL

PJI (54) - Mais que font nos députés ? Une sociologie informatique du travail parlementaire

Auteur : Quentin BAERT
Encadrant universitaire : Samuel HYM
Encadrant : Etienne OLLION

janvier à juin 2015

Sommaire

Remerciements	3
Introduction	4
1 Objectifs du projet	5
2 Choix des outils	6
2.1 Le langage Scala	6
2.2 SBT (Scala Build Tool)	6
3 Récupération des comptes rendus intégraux	7
3.1 Organisation des PDFs sur le site de l'Assemblée nationale	7
3.2 Récupération des PDFs	9
4 Isolement des scrutins	10
4.1 Conversion des PDFs en fichiers textes	10
4.1.1 La librairie PDFBox	10
4.1.2 Procédé	10
4.2 Analyse des fichiers textes pour isoler les scrutins	11
5 Récupération des données des scrutins	12
5.1 Représentation objet d'un scrutin	12
5.2 Extraction des données	13
5.3 Nettoyage des données	14
5.4 Construction des bases de données	16
6 Résultats	17

Conclusion	19
Bibliographie	20

Remerciements

Merci à Samuel Hym et Etienne Ollion de m'avoir encadré lors de la réalisation de ce projet, merci également pour leur accessibilité, leur bienveillance et pour avoir répondu à chacune de mes questions.

Introduction

Le PJI (Projet individuel) se déroule dans le cadre de la première année de master informatique à l'Université de Lille 1. Le but est ici de développer un projet sur l'ensemble d'un semestre.

Ce rapport concerne le projet numéro 54, intitulé "Mais que font nos députés ? Une sociologie informatique du travail parlementaire" et encadré par messieurs Samuel Hym, enseignant chercheur au laboratoire CRISAL de Lille, et Etienne Ollion, chercheur CNRS au laboratoire SAGE de Strasbourg.

Le but de ce projet est d'exploiter les comptes rendus intégraux de l'Assemblée nationale française afin de constituer des bases de données qui contiennent les informations des scrutins qui y sont votés : numéro, date, sujet du scrutin ainsi que nom, prénom, parti et vote des députés participants au scrutin.

Ce rapport présentera l'ensemble des travaux effectués sur le projet ainsi que leurs résultats. Pour commencer, le choix des outils utilisés sera exposé et justifié. Nous aborderons ensuite la manière dont les comptes rendus de l'Assemblée nationale ont été récupérés. Puis, nous verrons comment ces comptes rendus ont été filtrés et mis en forme afin de pouvoir en exploiter les données. Enfin, nous verrons comment les données ont été extraites et nettoyées afin de créer les bases.

1 Objectifs du projet

Ce projet se place dans le cadre d'un projet collectif mené à l'ENS et à l'Université de Strasbourg qui a pour but de mieux connaître le fonctionnement des deux assemblées législatives française : l'Assemblée nationale et le Sénat.

Son objectif est de collecter l'ensemble des scrutins publics tenus à l'Assemblée nationale entre 1958 et 2002 et d'organiser leur contenu dans une base de données facilement consultable (sous forme d'un fichier CSV par exemple). On trouvera dans cette base des champs tels que la date du scrutin, la législature durant laquelle il a été voté, son numéro, son sujet, ainsi que des information sur chacun des votants.

Les données initiales doivent être extraites du texte de documents PDFs. Ces PDFs sont des journaux scannés pour la plupart, dont le texte est souvent organisé en plusieurs colonnes (voir ...). Il faut donc trouver un outil capable de prélever efficacement du texte des PDFs tout en conservant son ordre naturel. On en tirera ensuite les données nécessaires à la création de la base.

Les différentes étapes de réalisation sont donc de :

1. récupérer tous les PDFs des comptes rendus intégraux sur le site de l'Assemblée nationale,
2. trouver un moyen d'extraire le texte des PDFs afin de pouvoir travailler sur le contenu des comptes rendus,
3. filtrer le texte obtenu pour isoler les scrutins,
4. nettoyer les données afin qu'elles soient exploitables,
5. créer un ou plusieurs fichiers CSV à partir des données prélevées.

La mise en place et les résultats de ces différentes étapes sont présentés dans la suite de ce rapport.

2 Choix des outils

2.1 Le langage Scala

Scala est un langage de programmation multi-paradigme qui compile sur la Java Virtual Machine.

Ce langage est utilisé pour ce projet car il intègre complètement le paradigme de programmation orienté objet, ce qui permet de retrouver des réflexes développés grâce à la pratique du Java. Son système de type, statique et inféré, offre une facilité au débogage couplée à une syntaxe allégée. Scala intègre également complètement le paradigme de programmation fonctionnelle, permettant donc une gestion améliorée des collections grâce aux opérations comme `map`, `filter` ou `fold`. Enfin, comme Scala est un langage qui compile sur la JVM, il est possible d'utiliser les bibliothèques Java au sein d'un code écrit en Scala, ce qui a donné la possibilité d'utiliser la librairie PDFBox comme nous l'exposerons plus tard.

2.2 SBT (Scala Build Tool)

SBT ou Scala Build Tool est un outil open source qui aide à construire une application Scala (qui contient potentiellement des sources Java). Cet outil permet de compiler facilement un projet Scala, facilite l'intégration de bibliothèques extérieures ou de frameworks de tests. Il intègre également un interpréteur Scala. SBT se configure grâce à un fichier `build.sbt`. Une copie du fichier utilisé pour ce projet est fournie en annexe.

Un projet construit à l'aide de SBT propose une architecture particulière. L'architecture complète du projet est également fournie en annexe.

L'utilisation de SBT a permis de suivre le cycle de travail suivant : écrire le code, compiler les sources, démarrer l'interpréteur, créer les objets nécessaires grâce aux documents et lancer directement le traitement.

3 Récupération des comptes rendus intégraux de l'Assemblée nationale

La première étape du projet est de récupérer les comptes rendus intégraux de l'Assemblée nationale depuis son site internet (<http://archives.assemblee-nationale.fr>) pour en extraire les données par la suite. Les sources du code qui sert à répondre à cette problématique se trouvent dans le package `download`.

3.1 Organisation des PDFs sur le site de l'Assemblée nationale

Les PDFs à récupérer sont ceux des comptes rendus intégraux tenus entre 1958 et 2002. Cela représente une période qui s'étend de la première à la onzième législature.

Le site de l'Assemblée nationale est organisé de la manière suivante : une page d'index par législature, chaque page d'index pointe sur des pages de sessions dans lesquelles se trouvent les liens vers les PDFs. Par exemple pour la dixième législature, on observe l'arborescence suivante :

```
10/cri/index.asp
|-- 10/cri/10-1996-1997-ordinaire1.asp
    |-- 10/cri/1996-1997-ordinaire1/001.pdf
    |-- 10/cri/1996-1997-ordinaire1/002.pdf
    |-- ...
|-- 10/cri/10-1995-1996-ordinaire1.asp
|-- ...
|-- 10/cri/10-1992-1993-extraordinaire3.asp
```

Où `index.asp` est la page d'index de la dixième législature, `10/cri/10-1996-1997-ordinaire1.asp` un exemple d'URL de page de session et `10/cri/1996-1997-ordinaire1/001.pdf` un exemple d'URL de PDF.

Dans un premier temps, nous pensons à générer chacun des URLs des PDFs auto-

matiquement en intégrant la logique de l'arborescence. Cependant, nous remarquons rapidement que certaines irrégularités nous feraient passer à côté de PDFs. Il existe par exemple des PDFs nommés `14a.pdf` et dont on ne peut prévoir l'apparition dans l'arborescence.

Afin de récupérer tous les URLs sans exception, nous avons décidé de récupérer le code HTML des pages d'index et d'y isoler les URLs des pages de sessions, puis d'ouvrir ces pages de sessions pour y trouver les URLs des PDFs.

Les pages d'index possèdent toutes la même structure : `http://archives.assemblee-nationale.fr/<numéro législature>/cri/index.asp`. Leurs URLs sont donc générés automatiquement. Puis, nous utilisons l'objet `Scala Source` et sa méthode `fromURL` qui renvoie le code HTML de la page sous forme de chaîne de caractères.

Comme le montre la figure 1, les pages de sessions sont référencées par des liens dont le texte est "Compte rendu intégral". Afin de trouver les liens des pages de sessions, nous isolons dans le code HTML les lignes qui contiennent les sous chaînes `href` (attribut HTML qui correspond à un lien) et "Compte rendu intégral".

1994-1995			
Session ordinaire (du 3 octobre au 21 décembre 1994) • Compte rendu intégral	Session extraordinaire (du 22 décembre au 23 décembre 1994) • Compte rendu intégral	Session extraordinaire (du 9 janvier au 19 janvier 1995) • Compte rendu intégral	Session ordinaire (du 3 avril au 30 juin 1995) • Compte rendu intégral
Session extraordinaire (du 4 juillet au 29 juillet 1995) • Compte rendu intégral			

FIGURE 1 – Capture d'écran d'une page d'index

Puis, dans les pages de sessions, tous les liens sont en fait des URLs vers les PDFs. Nous les capturons donc en isolant les liens `href` qui contiennent une sous chaîne `.pdf` de chacune des pages de sessions.

Ce traitement est exécuté par l'objet `URLManager` qui fournit, à travers son champ `pdfURLs`, la liste des URLs de tous les PDFs.

Lundi 3 octobre 1994	Séance unique		
Mardi 4 octobre 1994	Séance unique		
Mercredi 5 octobre 1994	Première séance	Deuxième séance	
Jeudi 6 octobre 1994	Première séance	Deuxième séance	Troisième séance
Vendredi 7 octobre 1994	Première séance	Deuxième séance	

FIGURE 2 – Capture d’écran d’une page de session

3.2 Récupération des PDFs

Une fois tous ces URLs recensés, il est facile de tous les télécharger. L’objet `PDFDownloader` possède une méthode `downloadAll` qui permet de télécharger chacun des PDFs ou d’afficher un message en cas d’erreur de téléchargement. Cependant, étant donné que 8691 URLs ont été récupérés par l’objet `URLManager`, une autre méthode `downloadGroupNb` permet de télécharger les PDFs uniquement par groupe de 100 (ces groupes sont numérotés de 0 à 86). Dans notre cas, nous téléchargeons 1000 PDFs par jour, cela permet de rester courtois et de ne pas surcharger les serveurs de requêtes.

Une autre problématique est de garder une organisation locale proche de celle du site internet afin de pouvoir s’y retrouver au milieu de ces milliers de PDFs. Pour cela, l’URL du PDF est légèrement modifié pour obtenir son chemin local qui est du type : `cri/<législature>/<années>/<session>/<nom>.pdf`.

Après le téléchargement de ces quelques 8000 PDFs, tous les comptes rendus de l’Assemblée nationale de la première à la onzième législature se trouvent en local sur notre machine. Reste maintenant à trouver un moyen de filtrer et exploiter ces 17 giga octets de données.

4 Isolement des scrutins

Seule une partie des comptes-rendus comporte des données intéressantes pour ce projet. La prochaine étape est donc de trouver un moyen de filtrer les documents afin d'isoler ceux qui contiennent une ou plusieurs analyses de scrutins.

4.1 Conversion des PDFs en fichiers textes

Afin de pouvoir explorer et exploiter les données des PDFs, nous choisissons de transformer chacun d'eux en fichier texte. Pour cela, nous utilisons la librairie PDFBox (<https://pdfbox.apache.org/>).

4.1.1 La librairie PDFBox

PDFBox est une librairie Java open source qui permet de travailler avec des documents PDF. Elle autorise notamment la création, la manipulation et la possibilité d'extraire du contenu de fichiers PDFs.

L'utilisation de cette librairie est rendue possible grâce à la particularité de Scala de pouvoir utiliser des librairies Java. De plus, elle est facilement importée dans le projet grâce à la simplicité d'utilisation de SBT.

4.1.2 Procédé

L'objet `PDFConverter` du package `pdftotext` procède à la conversion d'un PDF en fichier texte.

Pour cela, les PDFs sont récupérés grâce à leur chemin local. Ensuite, à l'aide des classes `PDDocument` et `PDFTextStripper` de la librairie PDFBox, le texte du document est récupéré. Puis, pour désigner le futur fichier texte, un nouveau chemin local du type `critxt/<législature>/<années>/<session>/<nom>.txt` est créé en se référant au chemin du PDF. Enfin le contenu du PDF est écrit dans un nouveau fichier texte. L'objet `PDFConverter` contient une méthode `convertAll` qui permet d'exécuter ce traitement pour tous les PDFs téléchargés.

4.2 Analyse des fichiers textes pour isoler les scrutins

Tous les documents qui concernent au moins une analyse de scrutin (et donc les documents qui nous intéressent) possèdent un critère commun simple : ils ont tous, en fin de fichier, une partie annoncée par le titre : "Annexe au procès verbal". Ces fichiers sont isolés grâce à l'objet `VoteFilter` du package `textanalysis`.

Le traitement de l'objet `VoteFilter` se résume de la manière suivante : ce dernier ouvre chacun des fichiers textes pour en explorer le contenu. La phrase "Annexe au procès verbal" est recherchée dans le fichier à l'aide d'une expression régulière Java. Si la phrase est trouvée, un nouveau chemin du type `scrutin/<législature>/<années>/<session>/<nom>.txt` est donné au fichier. L'objet comporte une méthode `filterAllTxtFiles` qui isole tous les fichiers contenant l'analyse d'un ou plusieurs scrutins.

Enfin, pour séparer les analyses de scrutin au sein d'un même fichier, la classe `VoteSeparator` du package `textanalysis` est utilisée. Celle-ci permet, à partir du texte d'un compte rendu, d'isoler et de renvoyer les différentes parties de textes qui représentent les analyses des différents scrutins du document. Pour repérer ces parties de textes, on se réfère au titre "Analyse du scrutin" qui annonce l'analyse. Un exemple d'analyse de scrutin isolée grâce à une instance de `VoteSeparator` se trouve en annexe.

5 Récupération des données des scrutins

L'étape suivante consiste à explorer les analyses de scrutins afin d'en tirer les informations nécessaires à la création des bases de données.

5.1 Représentation objet d'un scrutin

Nous avons décidé de donner une représentation objet aux scrutins et à chacun de ses composants afin de pouvoir se détacher du texte le plus rapidement possible. Le texte ne sert ainsi qu'à construire les divers objets, les différents traitements (écriture des CSVs, vérification de la cohérence des informations récupérées) sont ensuite écrits dans ces objets et exécutés par ces derniers.

Toutes les classes qui servent à représenter un scrutin se trouvent dans le package `vote`.

Les décisions de vote

Les différentes décisions de vote sont représentées par les objets suivants :

- `For` pour un vote "pour",
- `Against` pour un vote "contre",
- `Abstention` pour une abstention de vote,
- `NonVoting` pour indiqué que le votant n'a pas pris part au vote.

Le fait de s'abstenir est différent du fait de ne pas prendre part au vote. Les seuls députés qui ne prennent pas part au vote sont ceux qui ont une position particulière lors du vote : président de l'Assemblée nationale ou président de la séance par exemple.

Les votants

Les votants sont représentés par la classe `Voter`. Une instance de `Voter` comporte le nom du votant, son prénom et son parti politique.

Dans les scrutins récupérés, tous les votants ne sont pas explicitement cités. On

sait par exemple que pour tel parti, 12 députés ont votés pour et 3 contre. C'est souvent le noms des 3 députés allant contre la majorité de leur parti qui sont donnés. Une seconde classe **AnonymousVoter** a donc été écrite afin de représenter les votants dont on ne connaît pas le nom. Cette classe étend la classe **Voter** en fixant le nom et le prénom du votant à **anonyme**.

Les scrutins

Les scrutins sont représentés par des instances de la classe **Vote**. Cette classe contient toutes les informations relatives au scrutin et qui serviront par la suite à créer la base de données, à savoir :

- le numéro de la législature pendant laquelle s'est tenu le vote,
- la date du vote,
- le numéro du scrutin au sein de sa législature,
- le sujet du scrutin,
- le nombre de votants,
- le nombre de votes exprimés,
- la majorité absolue,
- le nombre de votes pour,
- le nombre de votes contre,
- la décision finale, le scrutin a-t-il été adopté ou non,
- le détail du scrutin, une structure qui donne pour chaque votant sa décision de vote.

5.2 Extraction des données

Cette partie représente le coeur du projet. En effet, c'est ici que les données concrètes vont être récupérées depuis le texte de l'analyse du scrutin et stockées dans des objets de type **Vote** précédemment décrit. L'objet **VoteBuilder** du package **textanalysis** se charge de récupérer ces données.

La première étape consiste à analyser les fichiers textes afin de repérer la manière dont les données sont représentées dans les analyses de scrutin. Les expressions

régulières Java nous paraissent être le meilleur moyen pour repérer et récupérer les informations. En effet, grâce à la fonctionnalité de groupe, elles permettent d'isoler des données au milieu d'un pattern afin de simplement les récupérer par la suite, si le pattern est trouvé dans la zone de recherche. Les expressions régulières écrites pour ce projet se trouvent dans la classe utilitaire `PatternDictionnaire` du package `utils`. Elles sont également toutes fournies en annexe de ce rapport. On y trouve par exemple l'expression `\d{1,2}\s+\p{L}+\s+\d{4}` qui permet de reconnaître dans le texte une date au format français.

Ensuite, à partir du texte de l'analyse d'un scrutin, l'objet `VoteBuilder` applique chacune des expressions régulières afin de trouver les parties du texte associées et d'en extraire les données. Il donne également une valeur par défaut aux champs pour lesquels le pattern n'a pas été reconnu, permettant ainsi de retrouver les erreurs ou les cas anormaux.

Toujours grâce à l'aide de SBT et de son interpréteur Scala, les expressions régulières sont testées sur des bouts de textes qu'elles sont censées repérer. Elles sont donc affinées à chaque essai et sont considérées comme viables une fois qu'elles parsent avec succès un certain nombre d'exemples.

5.3 Nettoyage des données

La dernière étape avant la construction des bases consiste à nettoyer les données récupérées. Pour certains champs, cela se fait plutôt immédiatement ; pour le sujet du scrutin par exemple :

sur la motion de renvoi en commission, présentée par M. Fabius, du projet de loi, adopté par le Sénat, relatif à la détention provisoire.

Ici, il suffit de supprimer les retours à la ligne et de reconstruire les mots fractionnés en fin de ligne.

Cependant, il est plus laborieux de nettoyer et transcrire une liste de votants d'un parti.

Groupe R.P.R. (259) :

Contre : 49 membres du groupe, présents ou ayant délégué leur droit de vote.

Non-votants : MM. Jean de Gaulle (président de séance) et Philippe Séguin (président de l'Assemblée nationale).

Groupe U.D.F. (206) :

Pour : 16 membres du groupe, présents ou ayant délégué leur droit de vote.

Contre : 3. - MM. Jean-François Chossy, Pierre Hériaud et Jean-Paul Virapoullé.

Groupe socialiste (63) :

Contre : 2. - MM. Camille Darsières et Jean-Jacques Fil-leul.

On remarque ici que plusieurs corrections sont à apporter au texte pour avoir une liste de votants propre. Voici quelques exemples de traitements à exécuter :

- retenir le groupe et la décision de vote avant de traiter la liste de votants associés,
- retirer les sauts de lignes et tous les signes de ponctuation inutiles, soit tous les signes de ponctuation sauf les virgules (elles servent de séparateur et sont utiles pour retrouver les noms et prénoms des votants) et les tirets (ils servent aux noms composés),
- éliminer des mots dont on est certain qu'ils ne sont pas des noms de députés (assemblée, nationale, Gouvernement, président etc.)
- éliminer la plupart des mots qui ne commencent pas par une majuscule, tout en préservant les particules (voir **Jean de Gaulle** de l'extrait précédent). Parmi les mots qui commencent par une minuscule, il faut ne préserver que ceux qui se trouvent entre deux mots commençant par une majuscule.

Une fois que les noms et prénoms des votants sont correctement isolés, il faut aussi ajouter les anonymes qui feront partie de la liste de votants finale.

5.4 Construction des bases de données

Une fois que les objets `Vote` ont correctement été créés, la dernière étape du projet consiste à construire un fichier CSV qui sera notre base de données. Celle-ci doit avoir la structure suivante :

Législature	Date	Numéro	Intitulé	Groupe	Nom	Prénom	Vote
-------------	------	--------	----------	--------	-----	--------	------

La classe `Vote` possède une méthode `toCSV` qui donne le texte à inscrire dans un fichier CSV pour représenter ce scrutin. Pour obtenir une base qui correspond à un fichier, une législature complète ou même toutes les législatures, un objet `CSVBuilder` est ajouté au package `textanalysis`. Ce dernier possède des méthodes pour construire plusieurs types de CSV, par exemple sa méthode `buildAllCSVFor` construit une arborescence similaire à celle des scrutins au format texte mais avec un fichier CSV pour chaque fichier texte.

Un exemple de CSV généré à partir d'un fichier texte est joint en annexe.

6 Résultats

Comme dit précédemment, le but de ce projet est de générer une base de données qui contienne l'ensemble des scrutins des dix premières législatures de la cinquième République. Les étapes de récupération et d'isolement des scrutins sont fonctionnelles, pour l'ensemble des documents. Cependant, lors de la récupération des PDFs, nous constatons que la grande majorité d'entre eux sont en fait des scans de comptes rendus probablement tapés à la machine à écrire. Nous savons alors que la transformation en fichiers textes ne sera pas de très bonne qualité. À ce moment, l'objectif du projet est réduit à construire une base qui contient le plus de scrutins possible, en partant de la législature la plus récente.

Partant de cette nouvelle considération, le code de récupération et de transformation des données est écrit pour les documents de la dixième législature. Il s'agit de voir ensuite à quel point il s'étend au reste des documents. Au moment où ces lignes sont écrites, seule une partie de la dixième législature est correctement prise en compte par le code. Actuellement, les données d'une trentaine de fichiers, soit environ soixante-quinze scrutins, sont correctement converties en fichiers CSV.

La travail continue donc en étudiant chaque document au cas par cas afin d'affiner le code et de faire en sorte que les scrutins soient convenablement traités, tout en veillant à ne pas régresser sur les fichiers correctement parsés. Les erreurs sont souvent résolues en élargissant la tolérance des expressions régulières qui capturent les données mais également en affinant la manière dont les données sont nettoyées. Néanmoins, il faut parfois modifier directement le fichier texte pour qu'il soit correctement analysé. Des erreurs qui ne peuvent être contournées par le code existent dans certains fichiers. Ces erreurs sont sûrement dûes au passage du format PDF au format texte ou directement dûes à une erreur de frappe dans le compte rendu original, et nous ne pouvons agir sur ces deux aspects.

Il est fort probable que le code actuel puisse être étendu jusqu'à englober la totalité de la dixième législature. Pourtant la qualité des fichiers textes baisse rapidement et nous pensons qu'une partie du code, notamment la capture et le nettoyage des données, devra être réécrite lorsque nous nous intéresseront aux législatures plus anciennes. Il se peut même que les données soient tellement sales qu'il faille envisager

une autre solution que la traduction en fichier texte afin d'y avoir accès.

Conclusion

Bibliographie