Université de Montpellier

Faculté des Scicences

30 Place E.Bataillon, 34095 Montpellier

Année 2023-2024

Rapport de Projet

Détection automatique des fake news à partir de données textuelles

par

22010416 - Romain GALLERNE

22009176 - Morgan NAVEL

21809267 - Éric GILLES

22004340 - Richard PICOLE-OLLIVIER

Groupe G6

Responsable du module : M. Pascal Poncelet

1 Pré-traitements

Afin de mener à bien notre projet, nous avons commencé par penser à une série de **pré-traitements** la plus complète possible ainsi qu'un moyen pour nous de pouvoir choisir tour à tour d'utiliser telle série de pré-traitements sans avoir à les générer à chaque itération. Parmi les pré-traitements que nous souhaitons tester, nous retrouvons : le **retrait des stop-words**, le **retrait des chiffres**, le **retrait de la ponctuation**, la **tokenisation des points d'exclamation et d'interrogation** (puisqu'ils transmettent une sémantique forte, cela peut être intéressant d'après nous), la **lemmatisation** et la **racinisation**.

Nous avons également souhaité tester un pré-traitement supplémentaire. En effet, notre jeu de données est plutôt déséquilibré et nous devons donc retirer certaines données (**downsampling**) afin d'équilibrer celui-ci, pour ne pas **biaiser la classification**. Cela résulte en la perte d'une quantité de données précieuses pour le classifieur. Nous avons donc décidé d'**ajouter un jeu de données supplémentaire à notre dataset**. Il s'agit d'un jeu de données issu du même programme que celui de nos jeux de base, à la différence près que celui-ci est en allemand. Il nous a donc fallu **traduire** le jeu de données de l'allemand vers l'anglais conformément à notre jeu de base. Pour faire cela, nous utilisons la **librairie de Google Translate** et nous sauvegardons le jeu traduit dans un **data frame** spécifique après avoir renommé chacune des colonnes et classes conformément à notre jeu initial (par exemple, la colonne *True* deviendra *true*, *Partially True* devient *mixture*).

Maintenant que nous avons tous nos pré-traitements, nous allons générer tous les sous-ensembles possibles comportant ou pas chacun des pré-traitements que nous avons évoqués. Cela donne **32 séries** de pré-traitements possibles, nous allons sauvegarder chacune d'entre-elles dans un fichier identifié par une suite de valeurs booléennes. Ces valeurs ont la sémantique suivante : *1.DataSetAllemandUtilisé* ; *2.ChiffresRetirés*, *3.PointsExclamation&InterrogationTokenisés*, *4.StopWordsRetirés*, 5.*Lemmatisation&Racinisation*. Ainsi, le pré-traitement "*TTFTF*" correspond à un dataset comprenant le **jeu en allemand** où les chiffres ont été retirés ainsi que les **stop-words**.

Afin de nous familiariser davantage avec les méthodes du **machine learning**, nous avons souhaité traiter par nous-mêmes certaines **fonctionnalités**, pourtant disponibles à l'aide de **librairies**. C'est par exemple le cas de **word_tokenizer** que nous avons entièrement recodé à la main avec une gestion spécifique de la ponctuation et des différents **tokens**.

Dans ce projet, nous utilisons l'approche **TF-IDF** pour la vectorisation. Nous avions également songé à utiliser **Bag of Words (BoW)**, cependant **TF-IDF** nous a paru être plus pertinent. Nous justifions ce choix par le fait qu'il fournit une représentation plus informative des données, il tient compte de l'**importance de chaque terme** en calculant quels sont les mots les plus **discriminants** et **révélateurs**. À l'inverse, **BoW** assigne un poids plus conséquent à un mot sur-représenté dans notre corpus. Dans notre cas, plus un mot est présent dans nos données, moins il sera important car **non discriminant**. C'est donc pour toutes ces raisons que nous avons choisi d'utiliser **TF-IDF**.

2 Modèles baseline : Recherche des meilleurs classifieurs

Dans ce projet, il nous était demandé d'effectuer **trois classifications différentes**. Nous présentons ici la méthodologie commune aux trois classifications. Tout d'abord, nous allons trier notre **data frame** en ne conservant que les documents ayant les annotations qui nous intéressent *(respectivement true et false, puis (true+false) et other puis true, false, other et mixture)*. Pour commencer, nous allons créer nos premiers modèles **baseline**, sans réaliser **aucun pré-traitement** sur notre **data frame** (si ce n'est la tokénisation et l'équilibrage du jeu de données bien sûr). Le but est ici de se donner une première idée de la **qualité** du jeu de données et de la classification que nous pourrons établir, nous l'améliorerons par la suite. Nous réalisons donc une première classification avec les paramètres par défaut de chaque **classifieur**. Nous utilisons un **Kfold** de **3 splits**, il s'agit simplement d'une mesure de référence et nous améliorerons également cela par la suite.

Nous avons donc commencé par faire tourner notre modèle avec les classifieurs suivants : *SVM, MultinomialNB, XGBoost, RandomForest, KNN* et *DescisionTree*. Nous avons souhaité intégrer un classifieur supplémentaire : **XGboost**, car d'après nos recherches, celui-ci pouvait s'avérer particulièrement pertinent dans notre cas d'usage. Voici les résultats que nous obtenons sans pré-traitements avec les paramètres **par défaut** :

Par souci de lisibilité, nous n'afficherons pas ici les matrices de confusions de chaque classifieur, cependant les valeurs de celles-ci confirment les choix de classifieurs que nous allons expliciter.

TABLE 1 – Valeurs de performance pour la classification VRAI/FAUX

Classifieur	Accuracy	Recall True	Recall False	Precision True	Precision False	FMesure True	FMesure False
KNN	0.61995	0.71415	0.5216	0.60273	0.68886	0.63728	0.56349
CART	0.63182	0.60865	0.65531	0.63839	0.62581	0.62297	0.64001
RFO	0.71492	0.71645	0.71468	0.71504	0.71576	0.71515	0.71463
SVM	0.7399	0.74205	0.739	0.73954	0.741	0.74027	0.73948
MultiNB	0.69356	0.87146	0.51812	0.64345	0.80545	0.73924	0.62839
Xgboost	0.71613	0.71812	0.71487	0.71622	0.71733	0.71642	0.71532

TABLE 2 – Valeurs de performance pour la classification (VRAI+FAUX)/OTHER

Classifieur	Accuracy	Precision	Recall	FMesure
KNN	0.52712	0.53565	0.52712	0.49382
CART	0.55068	0.56295	0.55068	0.54733
RFO	0.61520	0.62747	0.61520	0.61414
SVM	0.62175	0.65367	0.62175	0.60992
MultiNB	0.59136	0.65195	0.59136	0.56986
Xgboost	0.59797	0.60157	0.59797	0.59878

TABLE 3 – Valeurs de performance pour la classification VRAI/FAUX/OTHER/MIXTURE

Classifieur	Accuracy	Precision	Recall	FMesure
KNN	0.34629	0.38682	0.34629	0.31601
CART	0.37332	0.38211	0.37332	0.37133
RFO	0.40541	0.42350	0.40541	0.39803
SVM	0.47307	0.51519	0.47307	0.47350
MultiNB	0.38174	0.58271	0.38174	0.36473
Xgboost	0.42222	0.43179	0.42222	0.41898

On remarque donc que certains classifieurs se démarquent pour chaque classification. Pour la première classification : **RFO**, **SVM** et **XGboost** ont les meilleures valeurs d'**accuracy**. On constate pour ces trois modèles que la **précision** est plutôt équilibrée, ce qui indique que le modèle n'a pas plus de facilité à détecter une classe plutôt qu'une autre, contrairement à **MultinomialNB** par exemple qui a une grande différence entre ses valeurs de précisions. On constate aussi que les valeurs de **rappel** sont équilibrées, contrairement à **KNN**qui semble rater beaucoup de valeurs négatives par rapport aux positives. Enfin, nous éliminons également **DescisionTree** *(CART)* car l'**accuracy** de celui-ci est trop faible.

Nous effectuons exactement la même analyse pour les deux classifications suivantes et nous relevons donc les classifieurs suivants : Pour la classification *(VRAI+FAUX)/OTHER* nous relevons **RFO**, **SVM** et **MultinomialNB**. Pour la dernière classification nous choisissons **RFO**, **SVM** et **XGBoost**.

3 Recherche des meilleurs pré-traitements et hyperparamètres

Nous allons à présent tenter d'identifier quels sont les **meilleurs pré-traitements** et **hyperparamètres** pour chacun des classifieurs que nous avons sélectionnés. Pour cela, nous allons utiliser notre **chaîne de pré-traitements** dont nous avons parlé en introduction. Nous avons donc **32 pré-traitements**, sur lesquels nous allons à chaque fois effectuer une recherche de meilleurs hyperparamètres. Il apparaît évident qu'il serait ici extrêmement long d'effectuer une recherche exhaustive de type **grid_search** par exemple. Nous allons donc procéder à une **recherche aléatoire** à l'aide de la fonction **RandomizedSearchCV**.

Notre recherche de paramètres est réalisée à l'aide d'une **cross-validation**. Nous avons fait le choix d'effectuer notre recherche sur l'ensemble du jeu de données, sans séparation d'un **jeu de test**. Nous justifions ce choix, car il s'agit ici simplement d'une recherche de paramètres. Une fois les hyperparamètres trouvés, nous entraînons de nouveau un modèle sur ces paramètres avec, cette fois-ci, un **jeu d'entraînement** puis une prédiction sur notre **jeu de test**. Ce choix nous permet d'utiliser un maximum de données pour la recherche de meilleurs paramètres et d'obtenir des résultats plus pertinents sans trop restreindre notre jeu de données. De plus, **RandomizedSearchCV** procède de lui-même à une **cross-validation**. **RandomizedSearchCV** n'étant pas exhaustif dans ces recherches, nous réaliserons tout de même une **recherche exhaustive grid_search** à la fin lors de la mise en place de la **pipeline**, mais un ensemble bien plus réduits.

Concernant les paramètres de notre cross-validation, à la fois pour l'entraînement et pour le **Kfold** utilisé lors du test, voici les paramètres que nous avons sélectionnés et qui nous semblaient les plus pertinents : **nb_split** de 5 afin d'obtenir 20 % de **jeu de test**, ce qui nous semble être une valeur souvent conseillée. Nous avons donc une taille d'entraînement de 80 % et de test de 20 %. Concernant notre **seed**, celle-ci est fixée à 30 pour tout le projet. Enfin, nous réalisons **20 itérations** sur **RandomizedSearchCV**. Il s'agit du nombre qui nous a paru être le meilleur compromis entre fiabilité et rapidité d'exécution.

TABLE 4 – Meilleurs paramètres et pré-traitements pour VRAI/FAUX

Pré-trait	Class	Paramètres	Accuracy	Precision	Recall	FMesure
TFFFT	SVM	ker :rbf, gamma :0.01, C :100	0.81777	0.81994	0.817767	0.81785
TFTFF	SVM	ker :rbf, gamma :0.01, C :100	0.80345	0.80468	0.80345	0.80359
TTTFT	SVM	ker :rbf, gamma :0.01, C :100	0.80269	0.80480	0.80270	0.80269
TTFFT	SVM	ker :rbf, gamma :0.01, C :100	0.79817	0.80080	0.79817	0.79828
TFFFF	SVM	ker :rbf, gamma :0.01, C :100	0.79366	0.79480	0.79366	0.79385

TABLE 5 – Meilleurs paramètres et pré-traitements pour (VRAI+FAUX)/OTHER

Pré-trait	Class	Paramètres	Accuracy	Precision	Recall	FMesure
TFTTF	SVM	ker :rbf, gamma :1, C :10	0.70059	0.72094	0.70059	0.69765
FFFTT	SVM	ker :sigmoid, gamma :0.1, C : 10	0.69972	0.71512	0.69972	0.69916
TFTTT	SVM	ker :rbf, gamma :1, C :10	0.69546	0.70805	0.69546	0.69452
FTTFT	SVM	ker :sigmoid, gamma :0.1, C :10	0.69260	0.71381	0.69260	0.68731
FTFFT	SVM	ker :sigmoid, gamma :0.1, C :10	0.68570	0.69280	0.68571	0.68386

TABLE 6 – Meilleurs paramètres et pré-traitements pour VRAI/FAUX/OTHER/MIXTURE

Pré-trait	Class	Paramètres	Accuracy	Precision	Recall	FMesure
TFTFF	SVM	ker :rbf, gamma :0.01, C :100	0.51603	0.51883	0.51603	0.51398
FTFTT	SVM	ker :rbf, gamma :1, C :10	0.51524	0.54035	0.51524	0.51486
TTTTT	SVM	ker :rbf, gamma :1, C :100	0.50858	0.52440	0.50858	0.50949
FTFFF	SVM	ker :linear, gamma :0.1, C :1	0.50843	0.53344	0.50843	0.50798
TTTFF	SVM	ker :rbf, gamma :0.01, C :100	0.50735	0.51495	0.50735	0.50774

Matrices de confusion du meilleur modèle de chaque classification en Annexe.

On remarque que la classification **VRAI/FAUX** obtient de très bons résultats. Les classes semblent **équilibrées** comme on peut le voir sur la **matrice de confusion**, ce qui indique que le modèle n'est pas **biaisé** pour une classe plutôt qu'une autre. La valeur d'**accuracy** est également plutôt bonne avec **82 %**.

La seconde classification, **(VRAI+FAUX)/OTHER** obtient de moins bons résultats. C'est assez logique au vu du type de classification. En effet, dans cette classification, on regroupe sous le même label deux types de documents assez différents : les **VRAI** et les **FAUX**, qu'on cherche à classer par rapport aux **OTHER**. C'est une classification bien plus compliquée, car les documents **VRAI** et **FAUX** sont assez différents entre eux et la

limite entre ceux-ci et les documents **OTHER** est bien plus difficile à déterminer dans ce cadre. Cette intuition est confirmée par la **matrice de confusion** sur laquelle on constate que les documents **OTHER** sont généralement mieux classés comparés aux documents **TRUE/FALSE** qui sont bien plus difficiles à classer, ce qui tire l'**accuracy** du modèle vers le bas. Il s'agit donc d'une classification assez **déséquilibrée** et **biaisée**, puisqu'une classe est bien plus dure à classer qu'une autre.

La dernière classification n'obtient quant à elle pas de très bons résultats. L'**accuracy** de celle-ci est assez basse. Cependant, on constate qu'il n'y a pas ici de déséquilibre particulier entre les classes, excepté pour deux classes : **OTHER** et **TRUE** qui semblent plus proches d'après ce qu'on peut lire sur la **matrice de confusion**. Cela pourrait expliquer les résultats de la **seconde classification** avec la classe **TRUE** étant trop proche de la classe **OTHER**.

Revenons à notre dernière classification. Il apparaît que celle-ci est assez **équilibrée** comme nous le disions, ainsi il ne semble pas exister de problèmes inhérents au jeu de donnée pour cette classification (excepté la similarité entre vrai et faux). Il y a également une autre raison pour laquelle ce modèle obtient des résultats bien plus faibles. Comme nous l'avons déjà expliqué, nous avons systématiquement **équilibré nos classes** avant chaque expérimentation. Pour cette dernière classification, nous avons donc dû réduire chaque classe jusqu'à qu'elle comporte le même nombre d'éléments que la classe la moins représentée. Malgré le **jeu de données supplémentaire** que nous avons ajouté, cela représente la perte d'une grande quantité de données. Cette classification a donc bien **moins de données d'entraînements** que les deux autres, ce qui explique également ses faibles résultats.

Une dernière cause est, d'après nous, le facteur de **traduction** du jeu de données en allemand. Celui-ci ne comportait pas les même **classes** et nous avons donc choisi de **reclasser** tout le document en transformant notamment les articles classés **Partially True** et **Partially False** en **MIXTURE**. Cette traduction peut, elle aussi, expliquer le faible score de cette classification, bien que l'ajout de ce jeu supplémentaire reste utile et pertinent au vu de nos résultats.

SVM semble être de loin le meilleur modèle à notre disposition, intéressons nous au fonctionnement des différents classifieurs pour en comprendre la raison. En effet, **SVM** trouve un **hyperplan** de séparation pour classifier les données et fonctionne bien pour

des espaces de **grande dimension** aux frontières **non linéaires**, comme c'est le cas dans notre corpus. **XGBoost** quant à lui est un algorithme basé sur des **arbres de décision** utile pour des classifications **non-linéaire** bien que moins optimisé pour cette tâche que **SVM**. **RandomForest** combine plusieurs **arbres de décision**, et est particulièrement utile pour des données complexes comme les nôtres, il obtient d'ailleurs un assez bon score. **DecisionTree** construit un arbre de décision basé sur des règles de décision simples, il est plus adapté pour des problèmes de classification plus simple. **MultinomialNB** est un modèle plus probabiliste, assez polyvalent mais pas le plus adapté à notre cas. Enfin, **KNN** classe les échantillons en fonction de leurs **voisins les plus proches**, il est efficace pour des jeux de données de petite taille mais n'est donc pas très performant ici.

Maintenant que nous avons déterminé notre meilleure combinaison de **pré-traitements**, **classifieurs** et **hyperparamètres**, nous pouvons créer une **pipeline** afin de créer un modèle que nous entraînons sur l'ensemble de nos données. Ici, nous réalisons finalement un **grid_search** exhaustif afin d'identifier les meilleurs **hyperparamètres** possibles, ceux précédent ayant été déterminé à l'aide de **RandomizedSearch**. Enfin, nous exporterons le modèle en format **pkl**.

4 Améliorations des résultats

4.1 Augmentation du jeu de données

L'un des principaux problèmes que nous avons rencontrés, en particulier sur la dernière classification, est le **manque de données**. Nous avons réussi à pallier à ce problème en partie en ajoutant un **jeu de données supplémentaire** traduit depuis l'allemand. Comme on peut le constater, presque toutes nos meilleures classifications utilisent ce jeu de données supplémentaire. il s'avère donc que ce jeu de données a grandement **permis d'améliorer nos résultats**. Il est donc très probable que si nous pouvions étendre encore davantage notre jeu de données, cela nous permettrait d'**améliorer** nos classifications, en particulier sur la dernière. Nous avions tenté de faire cela, en particulier en **ajoutant les jeux de données venant de la même source, mais d'années différentes**, malheureusement, nous avons eu du mal à avoir accès à ceux-ci et le temps nous a manqué pour relancer l'ensemble du processus, notamment à cause de certains classifieurs, en particulier **XGboost** que nous avons souhaité ajouter, mais qui s'est révélé extrêmement long à utiliser.

(3 jours entiers pour réaliser l'ensemble de la recherche de meilleurs paramètres en RandomizedSearch sur une classification)

4.2 Topic Modeling

Le **topic modeling** *(modélisation de thèmes)* est une technique statistique utilisée en **traitement automatique du langage naturel** *(TALN)* pour déterminer automatiquement les **thèmes** ou **sujets** récurrents dans un ensemble de documents textuels. Cette technique pourrait s'avérer particulièrement utile lors de la réduction des jeux de données pour l'équilibrage des classes (**downsampling**). En effet, il est possible que si certaines classes aient des thèmes très marqués, alors notre classifieur pourrait s'attarder sur ces différents sujets plutôt que la véracité des articles. On peut par exemple imaginer que si une majorité des articles de la classe **VRAI** traitent de **politique**, alors le classifieur pourrait classer **VRAI** tous les documents **politiques** par **biais**.

Pour visualiser cela, nous avons cherché à réaliser une **modélisation des thèmes** de notre jeu de données. Nous avons donc réalisé un **graphe des thèmes**. Ce graphe offre une représentation visuelle des relations entre les thèmes, mettant en évidence leur proximité ou leur éloignement en fonction de leur similarité. Nous remarquons ici, qu'en effet, notre jeu de données n'est pas équilibré en termes de sujet. La classe **FAUX**, par exemple, comporte des articles très centrés sur des sujets de société assez anodins et ne comportent que quelques articles qui se détachent beaucoup de ce thème. La classe **VRAI** à l'inverse comporte une grande part de sujets **médicaux** ou liés au **COVID-19**, des sujets très spécialisés donc et très peu de sujets de société divers comme la classe **FAUX** *(voir Annexe)*.

Bien que nous puissions connaître les sujets globaux de notre corpus de données, il apparaît très difficile de **classer les articles** en fonction de leur **sujet**, car cela requiert une **procédure de classification à part entière** qui est trop longue et trop coûteuse à mettre en place dans le temps imparti à ce projet.

5 Annexes

5.1 Matrice de confusion du meilleurs classifieur lors de la recherche des meilleurs prétraitements et hyperparamètres

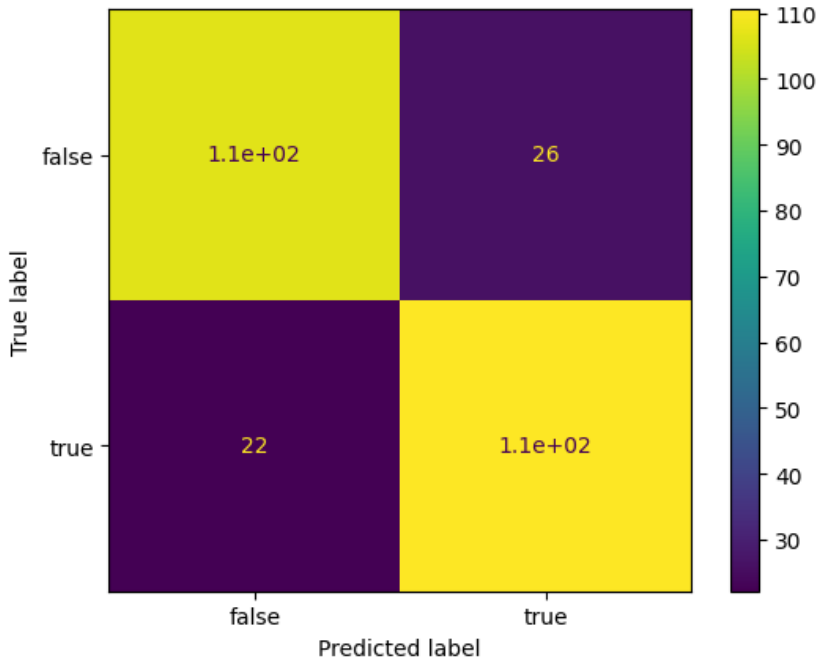

FIGURE 1 – Matrice de confusion pour la classification VRAI/FAUX

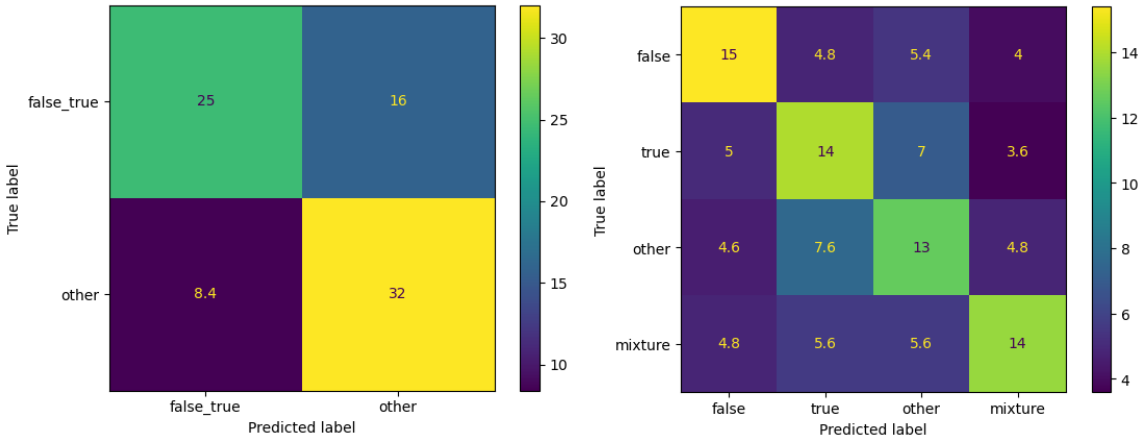

FIGURE 2 – Matrice de confusion pour la classification (VRAI+FAUX)/OTHER puis VRAI/FAUX/OTHER/MIXTURE

5.2 Sujets de notre corpus de textes

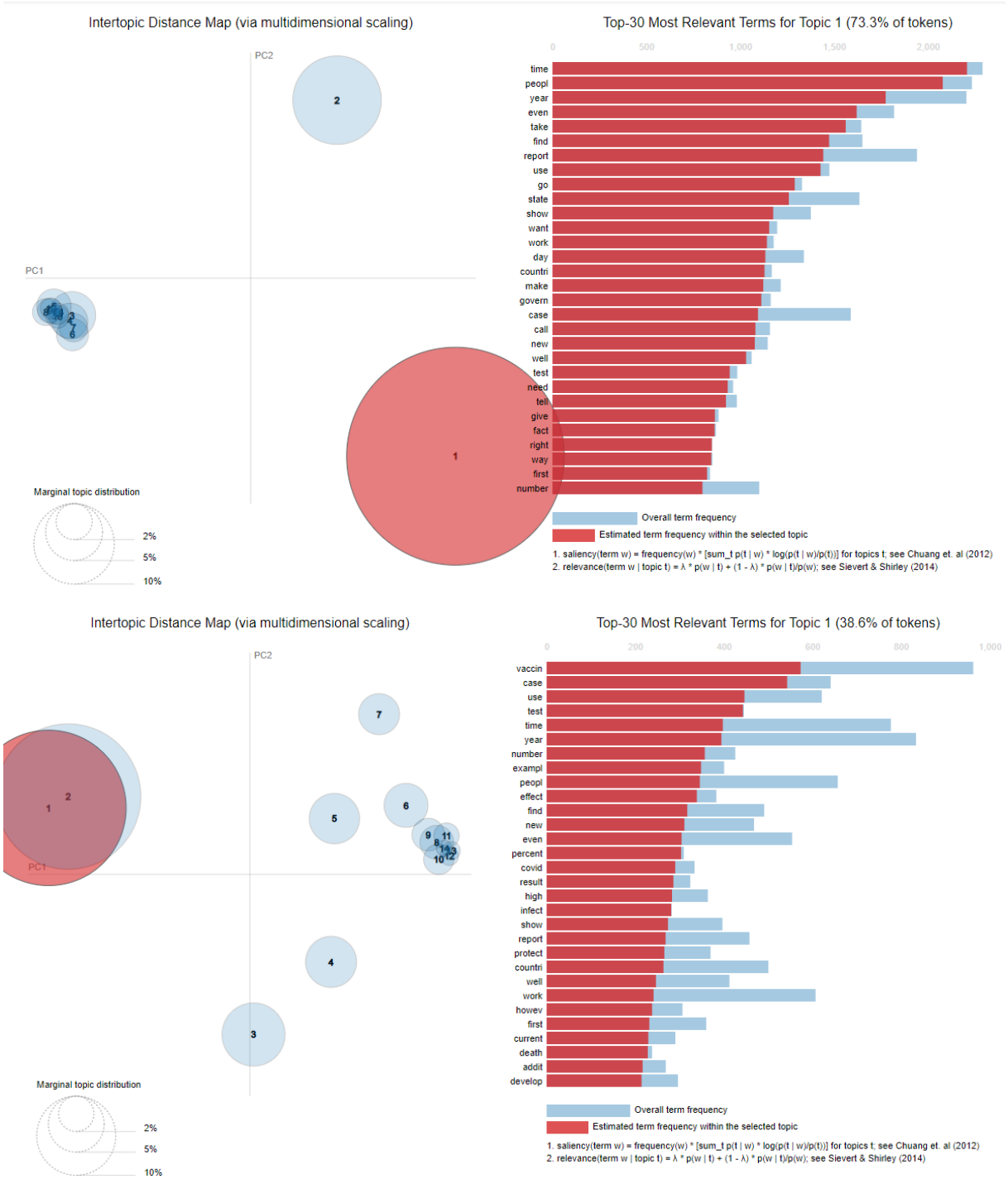

FIGURE 3 – Différents sujets de notre corpus de textes avec respectivement les classes *FAUX* et *VRAI*

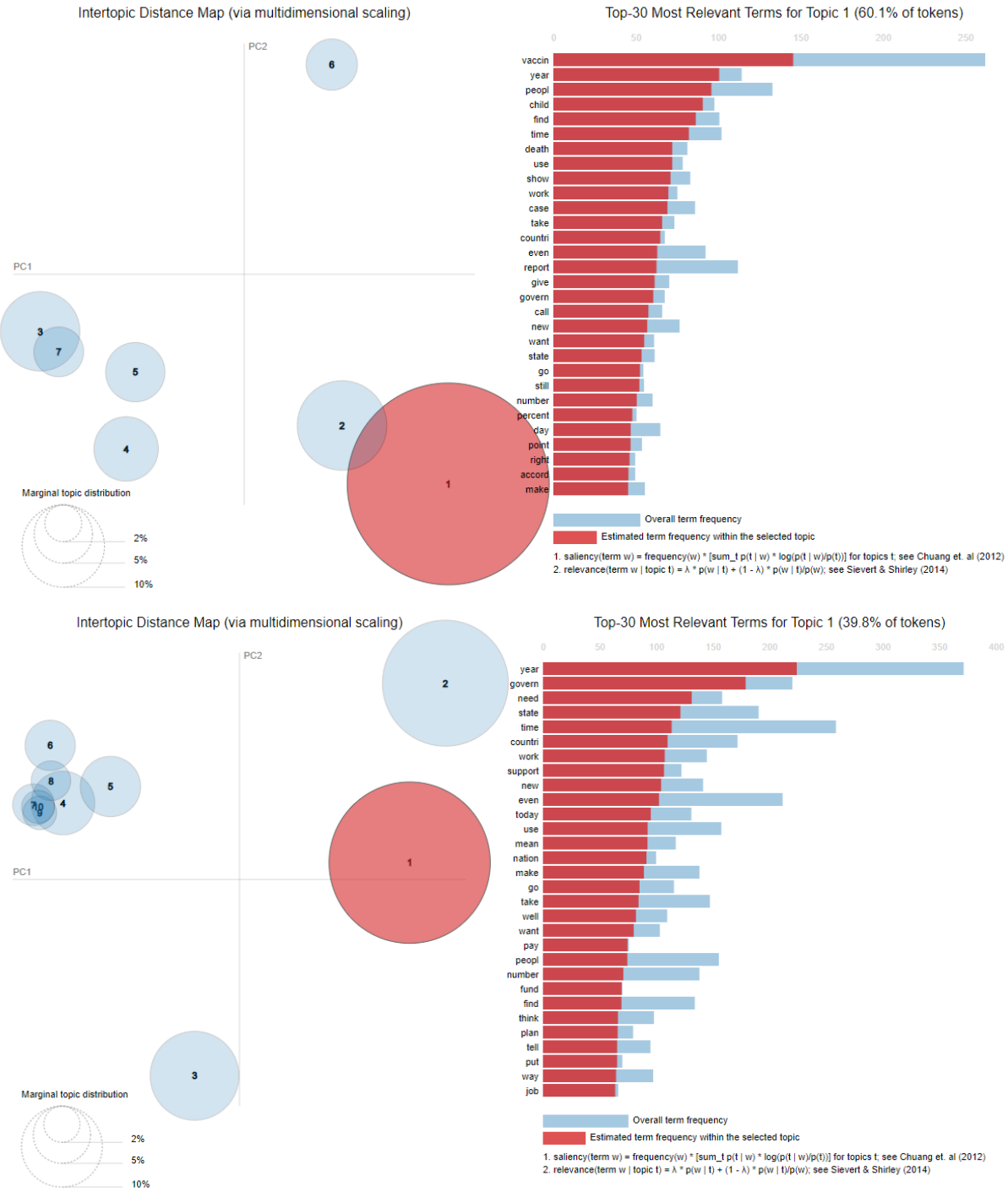

FIGURE 4 – Différents sujets de notre corpus de textes avec respectivement les classes *OTHER* et *MIXTURE*