PROJET DATA SCIENCE

Classification d'assertions selon leur valeurs de véracité (automatic fact-checking)

Claudia Restrepo-Ortiz ep. Auguet (n^oEtu: 21914554)*, Jérôme Reboul (n^oEtu: 18630129) and Xavier Grand (n^oEtu: 20064097)

Introduction

Les réseaux sociaux sont devenus un espace d'expression libre dans notre monde actuel. De nombreuses personnes les utilisent, et les figures politiques n'échappent pas à cette règle. Ainsi, de nombreuses assertions sont diffusées au plus large public sans aucun contrôle d'exactitude. La priorité est alors de produire du contenu (en référence à Marina Rolmann, "L'âge d'or du contenu", La drôle d'humeur de Marina Rollman, France Inter le 4 Mai 2020). Toutefois, méfions-nous de l'argument d'autorité, les figures politiques sont-elles toujours franches et honêtes ? Disent-elles toujours des vérités ? Sciemment ou non, mais cela sort du cadre de ce projet...

Il existe des sites de fact-checking (tels que www.politifact.org ou www.snopes.org) qui vérifient la véracité des informations disponibles sur la toile. C'est à partir de ces données, collectées par le LIRMM en collaboration avec plusieurs équipes de recherche européennes, que nous avons travaillé.

Description des données

Le jeu de données qui nous a été proposé provient de la base de données ClaimKG. Il s'agit d'un extrait de cette base qui contient 39218 assertions labélisées. Les labels, autrement dit l'attribution de la valeur de véracité de l'assertion, sont très divers. Il est nécessaire, pour réaliser une classification telle que demandée (c'est à dire de prédire si une assertion est vrai ou fausse, ou encore si elle est un mix d'informations vraies et fausses ou si elle est entièrement vrai ou fausse), de construire un label harmonisé. Nous pourrions créer un label qui ne contiendrait que les valeurs 'TRUE', 'FALSE, 'MIXTURE' et éventuellement 'OTHER'. Nous n'avons pas réussi à créer ce label correctement, ou d'une façon qui nous paraissait satisfaisante. Afin de pouvoir travailler, nous avons donc choisi de réaliser de nousveaux exports depuis la base ClaimKG à partir du portail Claims Search. De cette façon, nous avons créer deux jeux de données supplémentaires, un premier 'claimskg_result(TRUE_Vs_FALSE).csv' qui ne contient que des assertions labélisées 'TRUE' ou 'FALSE', vraies ou fausses afin de proposer un modèle de prédiction {VRAI} vs. {FAUX} ; et un second 'claimskg_result(TRUE-FALSE_Vs_MIXTURE).csv' qui ne contient ques des assertions 'TRUE', 'FALSE' ou 'MIXTURE', afin de proposer un modèle de prédiction {VRAI et FAUX} vs. {MIXTURE}. Il est à noter que le portail ne permette de ne récupérer que des jeux de données de 10000 assertions.

*Correspondence: claudia.restrepo-ortiz@etu.umontpellier.fr
Faculté des sciences, Université de Montpellier, Master SNS, parcours BCD., 34000 Montpellier, France
Full list of author information is available at the end of the article

Pré-traitement

Afin de pouvoir travailler sur ces données textuelles, il est nécessaire de réaliser un certains nombre de pré-traitements. En effet, les assertions ne peuvent-être utiliser dans les modèles de classification sous la forme de phrase. Dans cette partie, nous décrivons les différentes étapes de ce pré-traitement des assertions.

Suppression des caractères non ASCII

Les caractères non ASCII sont les caractères qui ne sont pas compris dans la liste des caractères utilisés dans notre algorithme, en l'occurence Unicode utf-8, c'est à dire tous les caractères qui ne sont pas codés sur 8 bits. Pour le traitements des données textuelles, nous ne pouvons conserver ces caractères, ainsi, le package 'unicodedata' grâce à sa fonction 'normalize' nous permet de normaliser le texte dans un format utf-8.

Suppression des contractions

La langue anglaise, comme la langue française, utilise un certains nombre de contractions. Il s'agit pas ex-

emple de la négation 'ne' qui peut, lorsqu'elle précède un mot commençant pas une voyelle, devenir 'n". De la même façon, en anglais, la négation 'not' devient "nt' à la fin d'un verbe. Afin de traiter des données textuelles, il est nécessaire d'harmoniser le texte et de supprimer ces contractions. Ainsi, le package 'contractions' nous permet de traiter ces cas de figure en remplaçant par exemple un 'does'nt' en 'does not'.

Tokenizing, ou segmentation des assertions

La tokenisation est une étape qui consiste à diviser les longues chaînes de texte en plus petits morceaux, ou jetons (tokens). Les plus grandes parties de texte peuvent être transformées en phrases, les phrases peuvent être transformées en mots, etc. Le traitement des données textuelles est généralement effectué après qu'un morceau de texte ait été correctement transformé en mots, en jeton. La transformation en jeton est également appelée segmentation de texte ou analyse lexicale. Parfois, la segmentation est utilisée pour désigner la décomposition d'un gros morceau de texte en morceaux plus grands que des mots (par exemple des paragraphes ou des phrases), tandis que la segmentation symbolique est réservée au processus de décomposition qui aboutit exclusivement à des mots.

Dans notre projet, nous considérons l'ensemble des assertions en une liste d'assertions, et chaque assertion est découpée en mots. Pour ce faire, nous utilisons le package NTLK, module *nltk.tokenize*.

Standardisation du texte:

Une des première étape du traitement des données textuelle est la normalisation. Elle fait généralement référence à une série de tâches connexes visant à harmoniser tous les textes : conversion de tous les textes en une seule et même casse (upper ou lower casse), suppression de la ponctuation, conversion des chiffres en leurs équivalents textuels, etc. La normalisation met tous les mots sur un pied d'égalité et permet de procéder au traitement de manière uniforme. Pour ce traitement, nous utilisons également le package NTLM.

Convertion en minuscules

L'utilisation des minuscules dans les données textuelles est l'une des formes les plus simples et les plus efficaces de prétraitement des textes. Applicable à la plupart des problèmes de text mining et de NLP (Natural Language Processing), elle peut aider dans les cas où le jeu de données n'est pas très important, et favorise considérablement la cohérence des résultats attendus.

Des variations dans la capitalisation du texte d'entrée (par exemple, "France" vs. "france") entraînent des résultats différents ou pas de résultats du tout. Cela est probablement dû au fait que le jeu de données présente des cas mixtes du mot, c'est à dire le même mot mais avec ou sans majuscule, et qu'il n'y a pas suffisamment d'occurences d'une des deux (ou plus) formes pour que le modèle apprenne efficacement les poids de la version la moins courante. Ce genre de problème est inévitable lorsque le jeu de données est assez restreint.

Remplacement des chiffres en lettres

Pour pouvoir être traités et vectorisés comme le reste du texte, les nombres en chiffres doivent être transformés dans leur version en toute lettre. De cette manière, le mot '2019' sera transformé en 'two thousand nineteen' et sera traité comme n'importe quel autre mot du texte.

Suppression de la ponctuation

Bien qu'elle ait une importance dans le langage humain, il est difficile d'en tenir compte dans notre analyse. En effet, il ne s'agit pas d'un item qui à une signification propre. C'est pourquoi nous la supprimons et considérons les accessions comme des phrases uniques, un ensemble de mots sans ponctuation.

Lemmatization

La lemmatization vise à éliminer les inflexions et à faire correspondre un mot à sa forme 'racine' (par exemple, "troubling" à "trouble"). Outre la modification des terminaisons, il est possible de transformer les mots dans la racine elle-même, par exemple le mot "car" serait attribué à "automobile", à cette fin un dictionnaire comme WordNet peut être utilisé, ou un dictionnaire créé ad-hoc. Pour réaliser la Lemmatisation de nos données, nous avons utilisé le module *nltk.stem*.

Stopwords, ou suppression des mots-vides de sens

Les"Stopwords" sont un ensemble de mots couramment utilisés dans une langue. On les appelle aussi des "mots vides" car ils n'apportent pas de valeur ajoutée aux systèmes de recherche et peuvent au contraire ajouter du bruit. Par exemple, "a", "the", "is", "are" sont des mots vides en anglais. Le principe de l'utilisation des stopwords est qu'en supprimant les mots peu informatifs du texte, nous pouvons nous concentrer sur les mots importants. La suppression de ces mots peut être efficace dans les systèmes de recherche et d'extraction de sujets, et permet d'obtenir des systèmes de classification plus rapides en réduisant le nombre de données considérées, ce qui contribue à maintenir une taille raisonnable des modèles. Le module nltk.corpus a été utilisé pour supprimer les stopwords de notres données, ce module contient un dictionnaire de stopwords de la langue anglaise.

Feature engineering, ou prise en compte des méta-données

Les méta-données, ou "features" que nous utilisons, influencent grandement le résultat. Aucun algorithme ne peut, à lui seul, compléter le "gain" d'information apportées par ces méta-données. Les scientifiques consacrent 80% de leur temps à la préparation des données ainsi que des méta-données, ce qui en démontre leur importance. La meilleure façon d'obtenir un bon traitement des "features" est de pratiquer différentes combinaisons dans divers ensembles de données et d'observer leur effet sur la performance des modèles. Pour notre ensemble de donnés, nous avons décidé de choisir comme méta-données supplémentaires aux assertions, l'auteur et la source. lorsqu'il s'agit de l'analyse des textes, le facteur humain doit être pris en compte, les déclarations faites, en particulier les personnalités publiques, sont directement liées à leur vie sociale, publique et professionelle. La prise en compte de l'auteur des déclarations peut nous donner une idée de la véracité de celles-ci. De même, les sources qui les publies sont cruciales. En effet, la sensibilité des médias et les ressources dont ils disposent pour obtenir l'information peut avoir un impact sur la manière dont ils délivrent l'information. D'autre part, nous avons également pris en compte les valeurs manquantes. C'est l'un des problèmes les plus courants que nous pouvons rencontrer lorsque nous essayons de préparer nos données pour l'apprentissage. La raison de la perte de valeurs peut être une erreur humaine, une interruption du flux de données, des problèmes de confidentialité, etc. Ces valeurs manquantes affectent les performances des modèles d'apprentissage. C'est pourquoi nous avons décidé d'éliminer les tuples où les informations de l'auteur étaient inconnues, et nous avons examiné l'effet que cela avait sur la précision de nos modèles par rapoort aux jeux de données complets.

Classification et sélection de variables

La classification que nous réalisons est une méthode d'apprentissage supervisé, c'est à dire que les classes, ou catégories, qui définissent le jeu de données sont connues. Un classifieur utilise les données du jeu pour tirer des règles qui attribuent, qui prédit, la classe à chaque tuple. Dans notre cas, par exemple, les assertions vraies et fausses doivent être utilisées comme données d'entrainement. Lorsque le classifieur est correctement entrainé, il peut être utilisé pour détecter une assertion inconnue, et évaluer si elle est vraie ou non, c'est à dire lui attibuer une classe "Vrai" ou "Faux". Cependant, la classification des données textuelles est un réel challenge et le choix d'un bon classifieur n'est pas évident.

Pour nos jeux de données, une fois que le prétraitement et la vectorisation des données textuelles en données numériques ont été effectués, plusieurs tests de classification sont réalisés. Il existe de nombreux algorithmes de classification, mais il n'est pas possible de conclure *a priori* s'il en est un meilleur qu'un autre. Cela dépend de l'application et de la nature du données et des méta-données disponibles. Ainsi, nous avons développé une approche empirique qui consiste à tester en comparer plusieurs classifieurs pour déterminer quel modèle est le plus adapté à nos données, lequel est le plus prédictif.

Les différents classifieurs testés

k-Nearest Neighbor

Le classifieur '*k-Nearest Neighbor*' (KNN) est un algorithme d'apprentissage lent qui stocke toutes les instances correspondant aux données d'entraînement dans un espace à n dimensions, n étant le nombre de méta-données distinctes. Lorsqu'une donnée inconnue doit être classée, le classifieur analyse un nombre k d'instances enregistrées, du jeu de données d'entrainenement, les plus proche (les voisins les plus proches) et lui attribue la classe la plus fréquente, la moyenne des k voisins les plus proches. Ce classifieur est robuste aux données brutées puisqu'il considère la moyenne des voisins les plus proches. Il peut donc convenir pour l'analyse de nos jeux de données textuelles. Le module KNeighborsClassifier de la librairie scikit learn a été utilisé pour ce classifieur.

Support vector machines

Les classifieurs '*Support vector machines*' et '*Linear Support vector machines*' (SVM et LSVC) permet, en plus d'effectuer une classification linéaire, d'effectuer efficacement une classification non linéaire en utilisant ce que l'on appelle l'astuce du noyau, ou 'kernel tricks' mapping implicitement leurs entrées dans des espaces de caractéristiques à haute dimension. Les avantages de ce classifieur est l'efficacité sur de grands jeux de données, même si le nombre de dimensions est supérieur au nombre d'échantillons, il utilise un sous-ensemble de points d'entraînement dans la fonction de décision, il est donc également efficace en mémoire. Toutefois, il ne fourni pas directement d'estimation de la probabilité, celle-ci est calculée par une coûteuse validation croisée. Nous avons testé à la fois le modèle linéaire ($LinearSVC$) et le modèle non linéaire (SVM), les fonctions linearSVC et SVC de la librairie scikit learn.

Decision Tree

Le classifieur '*Decision Tree*' (DTC) construit des modèles de classification ou de régression sous la forme

d'une structure arborescente, un arbre de décision. Elle utilise un ensemble de règles qui s'excluent mutuellement et qui sont exhaustives pour la classification. Les règles sont apprises séquentiellement à l'aide des données d'entrainement. Chaque fois qu'une règle est apprise, les tuples couverts par les règles sont retirés. Le module *DecisionTreeClassifier* de la librairie scikit learn a été utilisé pour ce classifieur. Toutefois, il convient de noter qu'un arbre de décision peut facilement être sur-entrainé, ou '*over-fitted*', générant trop de branches et pouvant refléter des anomalies dues au bruit ou à des valeurs aberrantes. Ceci est répété plusieurs fois en utilisant des sélections aléatoires de features et d'échantillons. Pour palier à cela, le paramètre '*random_state*' permet de contrôler ces choix aléatoires.

Gaussian Naive Bayes
Le classifieur *Naive Bayes* est un classifieur probabiliste qui repose sur l'hypothèse selon laquelle les attributs sont indépendants. Il s'agit un algorithme relativement simple à mettre en œuvre et qui donne de bons résultats pour de nombreux cas de figures. Il peut être facilement adapté à de grands jeux de données puisqu'il prend un temps linéaire contrairement à de nombreux autres types de classifieurs. Toutefois, lorsque la probabilité conditionnelle est nulle pour un attribut particulier, ce classifieur ne permet pas de faire une prédiction valable. C'est pourquoi le prétraitement des données textuelles est extrêmement important. Pour notre cas, nous avons utilisé le modèle le plus courant, le '*Gaussian Naive Bayes*' pour lequel la probabilité des feautures est censée être gaussienne.

Multi-layer Perceptron
Le classifieur *Multi-layer Perceptron* est une classe de réseau neuronal artificiel. Ce modèle de classifieur optimise la fonction de perte de registre en utilisant l'algorithme du gradient stochastique. Contrairement à d'autres algorithmes de classification tels que le*SVM* ou *Naives Bayes*, le *MLPClassifier* s'appuie sur un réseau neuronal pour effectuer la classification.

Logistic Regression
Le classifieur '*Logistic Regression*' est largement utilisé en raison de sa remarquable capacité à détecter les outliers. Il utilise une équation de régression linéaire pour produire des sorties binaires discrètes, tout en construisant des modèles discriminatoires de la même façon que le *SVM* et le *MLPerceptron*. Ce classifieur possède plusieurs paramètres qui permettent de sélectionner sa polyvalence au moment de l'analyse en fonction des données d'entrée. C'est le cas du paramètre *multi-classe* qui nous permet référencer une distribution binaire ou multimodale. Dans notre cas, ce paramètre a été défini comme 'multimodal', car même si nos données sont binaires, s'ajuste sur l'ensemble de la distribution de probabilité. Les autres paramètres sont *random_state*, *solver* qui spécifie l'utilisation de l'algorithme, dans notre cas l'utilisation de l'approximation multimodale limite à des algorithmes spécifiques dans ce paramètre, donc notre paramètre *solver* est référencé avec le methode *newton-cg*, qui est une optimitation non-linéaire.

Perceptron
Le classifieur '*Perceptron*' est un classifieur linéaire qui pondère tous les attributs. Cette pondération est appliqué à une fonction d'activation pour obtenir la valeur ajustée de chaque attribut. Ensuite, la classifiaction se fait grâce à la génération d'un réseau neuronal. C'est l'exemple de réseau neuronal le plus simple.

Random forests
Le classifieur '*Random forests*' est un algorithme d'apprentissage supervisé. Il peut être utilisé à la fois pour la classification et la régression. C'est également l'algorithme le plus flexible et le plus facile à utiliser. Une forêt est composée d'arbres, plus il y a d'arbres, plus une forêt est robuste. Les forêts aléatoires créent des arbres de décision sur des échantillons de données sélectionnés au hasard, on obtient alors des prévisions pour chaque arbre et on sélectionne la meilleure solution. Ce classifieur fournit également un bon indicateur de l'importance des attributs.

Validation Croisée et Entrainement

La validation croisée est un outil essentiel dans le traitement des données, que nous permet de mieux utiliser nos données. Lorsque nous construisons un modèle d'apprentissage, nous divisons nos données en un jeu d'entrainement (train) pour entraîner le modèle et un jeu de validation (test), "inconnu" du modèle entrainé. L'approche classique consiste à effectuer une simple division des données, une répartition d'un certains pourcentage dans le jeu d'entrainement et le reste dans le jeu de validation. Dans notre cas, nous avons utilisé 70% des données pour le jeu d'entrainement et 30% pour le jeu de validation.
D'autre part nous avons réalisé une validation croisée. Le principe est de réaliser des divisions (appelées Folds) différentes et répétées des jeux de données pour l'entrainement et la validation. Dans notre cas, nous avons réalisé des divisions par 10 des jeux (K = 10). Ainsi, le jeu de données est découpé en 10 parties, dont 9 constituent le jeu d'entrainement, et 1 le jeu de validation ; et cela est répété pour toutes les combinaisons de ce découpage. Nous obtenons ainsi une moyenne d'accuracy pour chaque modèle et non pas une valeur unique.

Analyse et discussions des résultats

Description des résultats

Nous avons utilisé plusieurs jeux de données différents sur lesquels nous avons entrainé les différents classifiers. Tel que proposé pour ce projet, nous avons créé deux jeux de données à partir de la base de données ClaimsKG. La description de ces jeux est faites en partie description des données en début de rapport.

Le jeu de données VRAIvs.FAUX

Ce jeu de données consiste en des assertions labélisées soit vraies soit fausses. La figure 1 résume les valeurs d'accuracy, de précision, des différents classifieurs entrainés sur ce jeu filtré dont l'auteur de toutes les assertions est connu. Ainsi, de manière empirique nous pouvons voir que le classifieur '*Logistic Regression*' est le plus précis avec une accuracy de 70% en moyenne. Le moins précis est le classifieur '*Perceptron*' avec une accuracy de 62%.

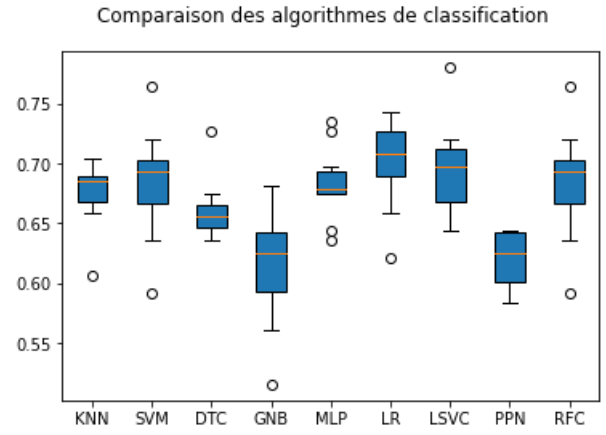

Figure 1 Figure 1 : Histogramme des accuracy des différents modèles de classifications testés pour le jeu de données 'claimskg_result(TRUE_Vs_FALSE).csv' sans les tuples dont les auteurs sont inconnus ('UNKOWN').
KNN : 'K-Nearest Neighbors', SVM : 'Support Vector Machine', DTC : 'Decision Tree Classifier', GNB : 'Gaussian Naive Bayes', MLP : Multi-layer 'Perceptron classifier', LR : 'Logistic Regression', LSVC : 'Linear Support Vector Classification', PPN : 'Perceptron', RFC : 'Random Forest Classifier'.

Puis nous avons réalisé l'entrainement des différents classifieurs sur ce jeu de données en considérant toutes les assertions, c'est à dire celles dont l'auteur est connu ou non. La figure 2 en présente les valeurs d'accuracy. Comme précédemment, le classifieur '*Logistic Regression*' donne les meilleurs résultats en moyenne, soit 74%. Et le classifieur '*Gaussian Naive Bayes*' obtient les moins précis avec 64% en moyenne.
Ainsi nous pouvons constater que le fait d'ajouter les assertions dont l'auteur est inconnu améliore les résultats des classifieurs. La raison à cette amélioration peut venir de la taille du jeu de données. En effet, lorsque le jeu de données est filtré sur les auteurs inconnus, il est plus petit. Ainsi l'entrainement des classifieur se fait sur un jeu restreint. De plus, nous pouvons nous poser la question de l'impact de l'anonymité des assertions sur leur véracité, cela pourrait expliquer le gain que nous avons.

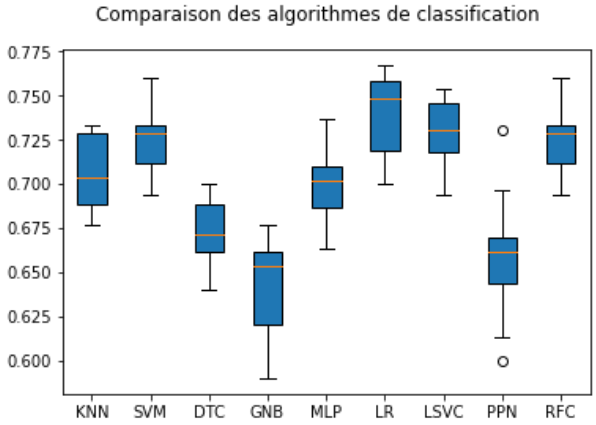

Figure 2 Figure 2 : Histogramme des accuracy des différents modèles de classifications testés pour le jeu de données 'claimskg_result(TRUE_Vs_FALSE).csv'.
KNN : 'K-Nearest Neighbors', SVM : 'Support Vector Machine', DTC : 'Decision Tree Classifier', GNB : 'Gaussian Naive Bayes', MLP : Multi-layer 'Perceptron classifier', LR : 'Logistic Regression', LSVC : 'Linear Support Vector Classification', PPN : 'Perceptron', RFC : 'Random Forest Classifier'.

Le jeu de données VRAI et FAUXvs.MIXTURE

Ce jeu de données consiste en une collection d'assertions qui sont labélisées soit vraies, soit fausses , soit en partie vraie seulement (MIXTURE). Nous avons créé deux classes pour les classifieurs, la classe VRAI et FAUX et la classe MIXTURE. Le but étant de déterminer si une assertion est soit totalement vraie soit totalement fausse, sans distinguer celles qui sont vraies de celles qui sont fausses, ou si cette assertion est en partie vraie, c'est à dire de la classe MIXTURE. Ainsi la difficultée est d'avoir un classifieur qui classe ensemble des assertions de véracités opposées (vrai/faux) et qui les sépare des assertions partiellement vraies. La figure 3 présente les résultats obtenus pour le jeu de données sans les assertions dont l'auteur est inconnu. Comme pour le jeu de données précédent, le classifieur '*Logistic Regression*' obtient le meilleur résultat avec une accuracy de 59.5%, et le classifieur '*Gaussian Naive Bayes*' qui est le moins précis avec une accuracy de 52.2%.

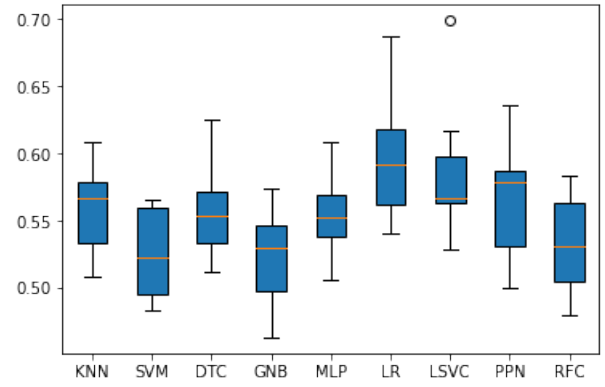

Figure 3 Figure 3 : Histogramme des accuracy des différents modèles de classifications testés pour le jeu de données 'claimskg_result(TRUE-FALSE_Vs_MIXTURE).csv' sans les tuples dont les auteurs sont inconnus ('UNKOWN'). KNN : 'K-Nearest Neighbors', SVM : 'Support Vector Machine', DTC : 'Decision Tree Classifier', GNB : 'Gaussian Naive Bayes', MLP : Multi-layer 'Perceptron classifier', LR : 'Logistic Regression', LSVC : 'Linear Support Vector Classification', PPN : 'Perceptron', RFC : 'Random Forest Classifier'.

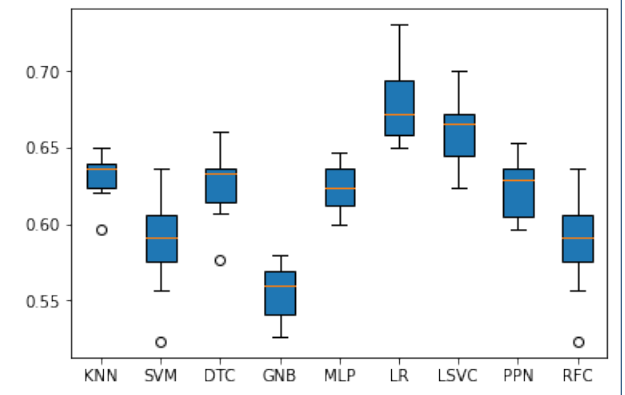

Figure 4 Figure 4 : Histogramme des accuracy des différents modèles de classifications testés pour le jeu de données 'claimskg_result(TRUE-FALSE_Vs_MIXTURE).csv' sans les tuples dont les auteurs sont inconnus ('UNKOWN'). KNN : 'K-Nearest Neighbors', SVM : 'Support Vector Machine', DTC : 'Decision Tree Classifier', GNB : 'Gaussian Naive Bayes', MLP : Multi-layer 'Perceptron classifier', LR : 'Logistic Regression', LSVC : 'Linear Support Vector Classification', PPN : 'Perceptron', RFC : 'Random Forest Classifier'.

Nous avons ensuite entrainé les classifieurs sur ce jeu de données mais en considérant toutes les assertions, dont les auteurs sont connus ou non. La figure 4 résume les résultats obtenus. Le classifieur '*Logistic Regression*' obtient le meilleur résultat avec une accuracy de 68%, et le classifieur '*Gaussian Naive Bayes*' qui est le moins précis avec une accuracy de 55%.

Ainsi, le classifieur '*Logistic Regression*' est celui qui obtient les meilleurs résultats quel que soit le jeu de données considéré parmis ce que nous avons testés. Nous constatons que l'utilisation des assertions dont l'auteur n'est pas connu permet d'améliorer la précision des classifieurs. Nous constatons également que la classification des assertions dans les classes VRAI et FAUX par rapport à la classe MIXTURE est plus difficile, comme nous le préssentions, possiblement en raison de l'opposition Vrai/Faux dans une même classe.

Discussions

Amélioration du pré-traitement

Nous avons réalisé un certains nombre de pré-traitements sur notre jeu de données avant de pouvoir entrainer les différents classifieurs. Ces pré-traitement sont classiques dans ce genre d'analyses textuelles. Notre liste n'est pas pour autant exhaustive. Nous aurions pu par exemple tester l'effet de la suppression des verbes sur les classifieurs. Une autre possibilité, aurait été de considérer des 'n-grams', c'est à dire des association de n-mots qui prennent un sens différents lorsqu'ils sont associés. Non pas que cela soit compliqué à mettre en oeuvre, nous l'avons en réalité tenté, mais les temps de calcul et l'espace mémoire nécessaire dépassaient la capacité de nos machines (peut-être y avait-il des solutions, mais nous ne les avons pas trouver à temps pour ce projet).

Utilisation des autre méta-données du jeu

Afin d'améliorer la précision de nos classifieurs, nous avons considérer les méta-données 'source' et 'author' des assertions dans nos modèles. Toutefois nous disposions d'autres features que nous aurions pu utiliser.

Ajustement des jeux d'entrainement et de test

Nous avons choisi d'utiliser une validation croisée pour entrainer et évaluer nos modèles de classification. La division en '10-folds' est un choix arbitraire. Nous pouvons nos poser la question sur l'impact d'une valeur de K différente sur la précision des classifieurs. Il faut garder à l'esprit que chercher à obtenir un classifieur trop performant peut conduire à son sur-entrainement. Il y a donc un compromis à accepter entre la précision du classifieur et sa polyvalence, sera-t-il performant seulement sur le jeu de données d'entrainement/validation ou le sera-t-il aussi sur un jeu différent.