28/04/2020

Fouille de textes

Classification automatiques avec Weka

Objectif : entraîner des classifieurs par apprentissage automatique et comparer les performances de différents algorithmes de classification sur une tâche de votre choix et un corpus que vous aurez constitué

Table des matières

Aurore LESSIEUX – Corentin VIALAR
ETUDIANTS EN M1 TAL – UNIVERSITE PARIS NANTERRE

Introduction

Dans les applications du Traitement Automatique des Langues, l'apprentissage automatique occupe une grande place, tant par son utilité dans le traitement du langage naturel que par les applications et les possibilités qu'il permet. La Fouille de Texte est une application majeure de ce domaine du TAL. Elle se décline en quatre tâches principales : Recherche d'Information (RI), Classification, Annotation et Extraction d'Information.

C'est la deuxième qui fait l'objet de ce projet, où nous avons sélectionné un corpus de textes que nous avons formaté à l'aide de programmes écrits à la main, puis traités à l'aide du logiciel Weka, dont le large panel d'algorithmes de classification automatique a permis d'établir pléthore de moyens de classer les textes en question dans des catégories que nous avions définies en amont.

Notre sujet d'étude est la classification d'ordinateurs en vente sur internet (ldlc et materiel.net), d'une part dans des catégories reflétant leur usage de prédilection, et d'autre part dans des classes correspondant à des fourchettes de prix

Le corpus choisi est constitué d'environ 600 descriptifs techniques d'ordinateurs, que nous avons réparti dans les catégories sus-citées. Il s'agit donc là de deux tâches de classification distinctes, opérées sur cet ensemble de textes.

À travers l'utilisation de Weka, nous avons cherché le meilleur algorithme de classification pour ces textes selon les catégories que nous avons imposées, sachant au départ que la distinction entre fourchettes de prix et celle entre usage seraient probablement très différentes dans leur traitement.

Composition du Corpus

Notre choix d'items à classifier s'est assez vite porté sur les ordinateurs – petit clin d'œil à nos études et à nos loisirs. Il nous a alors fallu décider des catégories de classification à mettre en œuvre. Pour ce faire, nous sommes souvenus qu'en prévision de ce master, nous avions tous deux fait l'acquisition de nouvelles machines plus performantes et plus adaptées à l'utilisation que nous allions en faire cette année. La première classification était donc toute trouvée : l'usage ; s'ensuivit une deuxième qui nous a particulièrement concernée dans la recherche de notre compagnon informatique en tant qu'étudiants : le prix. Entre ces deux idées, aucune n'a su gagner notre préférence sur l'autre. Nous avons donc choisi de traiter les deux. Venait à présent la question des catégories de chaque classification.

Pour l'usage, nous avons fait le choix d'établir deux catégories : gaming et bureautique. D'autres idées ont été envisagées dans la catégorisation des usages, telle que la catégorie design/graphique, mais n'ayant qu'une connaissance approximative de ce marché nous n'étions pas certains de trouver suffisamment de contenu de cet usage particulier pour le corpus, contrairement aux ordinateurs conçus spécifiquement pour le gaming ainsi que ceux fait pour un travail bureautique qui sont disponibles en grand nombre.

Pour le prix, nous souhaitions avoir 3 fourchettes de prix différentes. Les valeurs se sont imposées d'elle-même lorsque nous avons construit le corpus afin d'obtenir une classification un minimum homogène en termes de contenu textuelle par classe et en représentativité d'une échelle de début de gamme, milieu de gamme et haut de gamme. Les catégories de prix sont donc : 0, pour la fourchette de 0 à 700 €, 700 pour celle de 700 à 1400 €, et 1400 pour les prix allant de 1400 € à plus.

Concernant le contenu du corpus, nous sommes partis du postulat que le prix ou l'usage d'un ordinateur dépendait de ses composants. Le texte qui constitue notre corpus se compose de fiches techniques d'ordinateurs que l'on peut retrouver dans la description des produits sur les sites de vente de matériel électronique. De par ce choix, d'autres idées de classification auxquelles nous avions pensé étaient compromises, telle que la classification PC de bureau vs. PC portable. En effet, dans la fiche technique figure la taille de l'écran, ce qui aurait considérablement biaisé nos résultats, étant donné qu'il s'agit là d'un critère déterminant et toujours présent dans les fiches techniques des ordinateurs portables, mais absent de celles des ordinateurs de bureau.

Pour la collecte du corpus, nous avons fait le choix d'automatiser un maximum la procédure : Corentin a écrit un script qui extrait la fiche technique à partir des urls – explication dans la partie suivante. Il restait tout de même un travail manuel de collecte des urls dont Aurore c'est chargé. Les sites dont nous avons récupéré les urls de produit sont LDLC et materiel.net. L'idée d'utiliser un géant du marché comme la Fnac a été envisagée mais bien vite abandonnée en raison de problème d'extraction de donnée (connexion refusée malgré divers essais de commandes pour récupérer automatiquement les pages).

Le corpus final comporte :

	0-700€	700-1400€	1400€ +	gaming	bureautique
LDLC	61	80	84	92	92
Materiel.net	97	192	123	181	218
Total	158	272	207	273	318

Pour la classification des fiches techniques que nous avons faites, nous nous sommes basés sur les classifications faites par les sites – comme c'est le cas de materiel.net, ce qui explique le nombre de données plus important – ou la description du produit – pour LDLC. Dans la mesure du possible nous avons essayé d'intégrer autant que possible le même nombre d'ordinateurs de bureau et d'ordinateurs portables dans chaque catégorie.

Formatage des données

Une fois les URLs enregistrées dans des fichiers correspondants à chaque catégorie, nous avons utilisé un programme python (extract_desc.py) écrit par Corentin pour télécharger les pages (à travers la commande wget lancée depuis le programme python), puis extraire du fichier téléchargé la partie qui nous intéressait (c'est-à-dire le tableau récapitulatif des caractéristiques techniques de l'ordinateur en question).

Pour ce faire, le module lxml a été mis en œuvre, permettant de parcourir l'arborescence d'un fichier xml ou html et d'en extraire des données.
Pour la plupart des caractéristiques techniques, une seule valeur était donnée (exemples : « processeur : i5-9600K », ou encore « carte graphique : GeForce RTX 2080 Ti ») ; cependant, des sections comme « connectique avant / arrière » contenaient une liste de valeurs (« 3xusb 3.1, Jack 3mm, 2xusb 2.0, etc. ») enregistrées chacune dans une ligne de tableau séparée, sans information dans les balises permettant de déterminer à quelle caractéristique elles correspondaient. Il a donc fallu élaborer un algorithme adapté à la structure logique du tableau pour regrouper chaque liste en une seule chaîne de caractère avant de l'associer au label correspondant.

Pour chaque fichier, le programme python générait un dictionnaire contenant les couples label-valeur(s) mentionnées ci-dessus. Il écrivait ensuite ce dictionnaire dans un fichier sous forme csv (séparé par des virgules) grâce au module pandas.

Le fichier de sortie a donc cette forme :

```
Désignation, Lenovo V330-15IKB (81AX00J2FR)
Marque, LenovoLenovo
Modèle, 81AX00J2FR
Système d'exploitation, Windows 10Professionnel 64bits
Famille OS, Microsoft Windows 10
Langue de l'OS, Français
Système d'exploitation fourni, Oui
Marque processeur, Intel
Processeur, Intel Core i3
Plateforme (Proc.), Intel Kaby Lake
Type de processeur, Intel Core i3-8130U (Dual-Core 2.2GHz / 3.4GHz Turbo - Cache 4Mo)
Fréquence CPU, 2.2GHz
Nombre de core, 4
Taille de la mémoire, 4Go
Nombre de barrettes, 1
```

Nous avons tenu à conserver les labels (à gauche) comme tels, sans normalisation d'aucun type : pas de regroupement de labels sous un même nom (ex : Carte Graphique, GPU et Processeur Graphique auraient pu être remplacés par une appellation unique), et pas d'élimination des labels uniques ou rares (retrouvés pour seulement certains PCs). Nous avons considéré que les supprimer pourrait soit trop simplifier la tâche au classifieur, soi retirer de l'information qui

aurait pu lui être utile (la classification étant automatique, il est impossible de savoir précisément si un trait sera considéré comme utile ou non par l'algorithme).

De même, au sein des valeurs, nous avons fait le choix de ne pas simplifier les appellations de certains composants qui pouvaient être similaires : par exemple, les processeurs Intel se déclinent majoritairement sous trois gammes : i3 (entrée), i5 (milieu) et i7(haut de gamme ; encore rares et récents, les i9 sont le très haut de gamme). Cependant, selon la date de sortie, les performances et l'architecture du processeur, son appellation se décline sous 4 chiffres complémentaires et parfois une lettre (ex : i5-7600K). Malgré le nombre imposant de modèles, nous n'avons pas voulu simplifier ces désignations, car un processeur haut de gamme plus ancien peut avoir des performance comparables ou moindres qu'un processeur de gamme plus basse, mais plus récent ; de même, deux processeurs i5 différents peuvent servir deux usages (gaming vs bureautique), d'où le sens de cette distinction dans notre classification.

Pour finir, il nous a fallu normaliser la mise en forme de certaines valeurs :

- Conserver les chiffres et les nombres dans la vectorisation (sans quoi de nombreuses informations sont perdues et la classification perd de son efficacité) ;
- Retirer les espaces qui séparent chiffres et unités de mesure : ainsi, 2666 MHz devient 2666MHz. En effet, non seulement cet espace est absent de manière aléatoire dans une partie des fiches (ou même au sein d'une même fiche), mais les valeurs n'ont pas de sens sans leur unité de mesure.

Tous ces choix de formatage enfin appliqués au travers de scripts python (joints à ce fichier), nous avons pu passer l'ensemble du corpus au script de vectorisation, puis ouvrir le fichier de sortie avec Weka pour commencer nos expérimentations et nos mesures.

Classification avec Weka

Parmi les différents algorithmes de classification par apprentissage automatique proposés par Weka, nous en avons sélectionné 11 sur la base de deux critères : la diversité de leurs méthodes de traitement, et les résultats qu'ils obtiennent. Nous avons donc testé chaque algorithme de chaque catégorie de classifieurs (sauf ceux des catégories « misc » et « meta ») pour chacune de nos deux classifications ; et nous avons retenu ici ceux qui ont obtenu les meilleurs résultats.

Ensuite, pour chacun des algorithmes sélectionnés, nous avons testé leur performance selon deux méthodes d'entrainement : cross-validation/validation croisée (avec un paramètre de 10), et percentage split (pour de meilleurs résultats, la division entraînement-test était de 80-20).
La validation croisée consiste à diviser le corpus en segment égaux (de taille spécifiée en paramètre, dans notre cas, 10%), à entraîner le modèle sur tous les segments sauf un, qui sera le segment de test, puis de recommencer avec chacun des autres segments.
Le percentage split prend en paramètre un pourcentage selon lequel il divisera le corpus entre corpus d'entraînement et corpus de test.
C'est le plus souvent la validation croisée qui atteint les scores les plus élevés pour la classification par prix, malgré des temps de traitement légèrement plus longs, tandis que le percentage-split l'emporte pour ce qui est de la classification par usage.

Note : Dans les tableaux qui suivent, un code couleur a été utilisé pour mettre en valeur les meilleurs résultats : des nuances de vert d'intensité croissante soulignent les 3 meilleurs résultats d'un tableau, sur la base des F-mesures, précisions et rappels obtenus. Un vert très intense indique un résultat proche de la perfection. Certaines cases sont marquées de rouge ou d'orange pour signaler une performance mauvaise ou médiocre.
De plus, on a ajouté les matrices de confusion de chaque classifieur : celle du meilleur de chaque catégorie est présentée sous le tableau concerné, mais la totalité de ces matrices est accessible dans l'archive, sous ./capt/.

Lazy :

Ces algorithmes sont du type « k plus proches voisins », où un document est catégorisé en fonction de sa proximité à d'autres (voisins) déjà classés dans une représentation multidimensionnelle du corpus.

Classification par Prix :

Les tests de 2 algorithmes ont été retenus ici : IBk sous deux formes : IBk-1 (1-plus proche voisin), IBk-3 (3 plus proches voisins) et LWL-7, dont nous ne nous attendions pas qu'il

obtînt de meilleurs résultats que IBk (qui avait été présenté en cours), sur la classification par prix. D'ailleurs, ses résultats étaient moindres avec un nombre différent de voisins. Les valeurs de k retenues (3 et 7) sont celles qui ont produit les résultats les plus élevés avec leurs modèles respectifs.

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
IBk-1	0.816	0.816	0.815	<1s	0.780	0.796	0.782	0.29s
IBk-3	0.819	0.820	0.818	<1s	0.815	0.825	0.815	0.44s
LWL-7	0.856	0.858	0.855	<1s	0.829	0.840	0.831	0.6s

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 133   0  20 |   a = 0
   1 168  32 |   b = 1400
  21  16 231 |   c = 700
```

Classification par Usage :

Dans cette partie n'ont été retenus que les algorithmes de ce type qui obtenaient les meilleurs résultats : IBk-1 et LWL-1. Lorsque k variait entre 1 et 3, les résultats étaient identiques, mais au-delà, ils se détérioraient. On remarque d'emblée des résultats quasi parfaits (moins de 10% d'erreur), peut-être dus en partie à la binarité de la classification.

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
IBk-1	0.986	0.986	0.986	<1s	0.991	0.992	0.991	0.23s
LWL-1	0.986	0.986	0.986	<1s	0.991	0.992	0.991	0.26s

```
=== Confusion Matrix ===

  a  b   <-- classified as
 61  0 |  a = bureautique
  1 54 |  b = gaming
```

Arbres de décision :

Les arbres de décisions permettent d'attribuer une classe à un document en se basant sur des attributs discriminants dans les textes, examinés en plusieurs étapes. À chaque étape, le programme cherche à départager les textes selon l'attribut optimal, puis recommence l'opération en sélectionnant un attribut parmi ceux qui restent, si la classe n'a pas été déterminée.

Nous avons testé 3 classifieurs de ce type : J48, LMT et RandomForest (RF ci-après ; qui a la particularité de générer de nombreux arbres et de les comparer entre eux).

Classification par Prix :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
J48	0.844	0.844	0.844	1.95s	0.796	0.805	0.798	<1s
LMT	0.881	0.882	0.881	32.39s	0.820	0.830	0.823	0.02s
RF	0.860	0.861	0.860	1.56s	0.831	0.844	0.831	1.59s

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 137   0  16 |   a = 0
   0 174  27 |   b = 1400
  15  16 237 |   c = 700
```

LMT produit indéniablement les meilleurs résultats, mais en un temps particulièrement long. RF, en comparaison, semble un bon compromis temps de traitement/qualité de classification.

En annexe : arbre de décision de J48 pour déterminer la classe de prix

Classification par Usage :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
J48	0.983	0.983	0.983	0.22s	0.991	0.992	0.991	<1s
LMT	0.991	0.991	0.991	8.23s	1	1	1	0.01s
RF	0.990	0.990	0.990	0.66s	1	1	1	0.64s

```
=== Confusion Matrix ===

  a  b   <-- classified as
 61  0 |  a = bureautique
  0 55 |  b = gaming
```

LMT est de nouveau nettement plus lent que les autres, mais se démarque par d'excellents résultats, cependant égalé par RF lorsqu'on divise le corpus selon un percentage split 80-20.

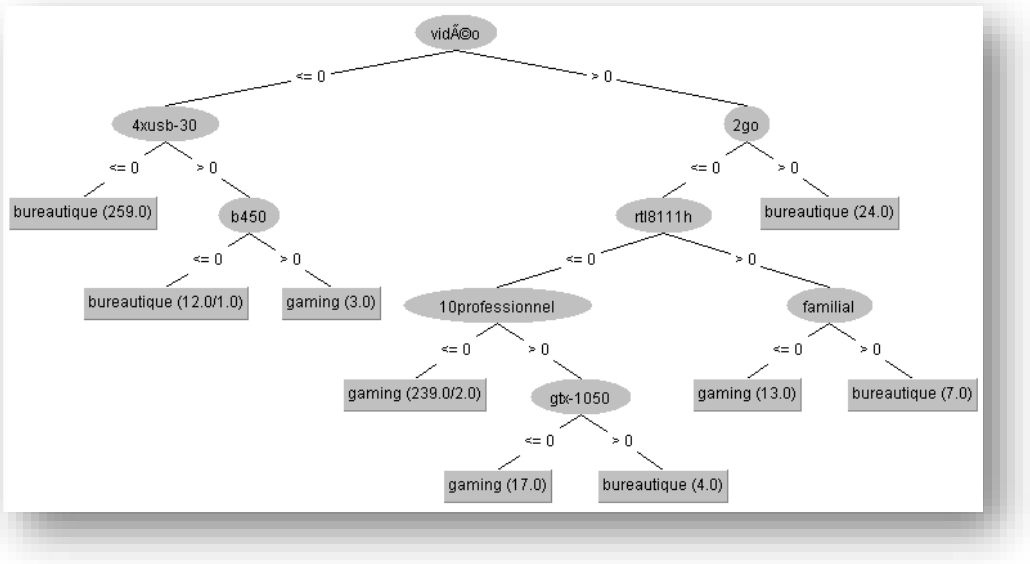


Fig1. : l'arbre de décision (plus complexe) de J48 pour le type d'utilisation.

Bayes Network et Naive Bayes:

Basé sur les probabilités conditionnelles (théorème de Bayes), cet algorithme détermine la classe d'un document en fonction des attributs qu'il y retrouve, et dans le cas de Naive Bayes, en faisant l'hypothèse que les attributs d'un document sont indépendants les uns des autres.

De par la littérature dont nous avions connaissance sur Naive Bayes, l'hypothèse que nous avions à ce stade était une supériorité de ce modèle sur Net.

Classification par Prix :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
Net	0.804	0.813	0.807	0.19s	0.778	0.798	0.782	0.32s
Naive	0.781	0.782	0.781	0.16s	0.717	0.718	0.718	0.08s

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 141   0  12 |   a = 0
   0 178  23 |   b = 1400
  23  62 183 |   c = 700
```

Notre hypothèse est cependant infirmée ici, même si d'à peine deux pourcent.

Classification par Usage :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
Net	0.870	0.875	0.870	0.16s	0.819	0.819	0.819	0.17s
Naive	0.924	0.926	0.924	0.07s	0.914	0.914	0.914	0.07s

```
=== Confusion Matrix ===

   a   b   <-- classified as
 276  31 |   a = bureautique
  13 258 |   b = gaming
```

Ici, Naive Bayes est à la hauteur de nos attentes, et avec un une marge plus nette.

Rule-based :

Cette famille d'algorithme crée elle-même ses règles de classification ; les méthodes varient selon l'algorithme.

ZeroR en est le plus simple exemple, et classe tous les documents dans la catégorie la plus fournie en documents, d'où ses résultats d'une faiblesse notoire. Nous l'avons mentionné ici uniquement à titre anecdotique.
JRip génère des règles indépendantes et se défait des règles peu ou pas utiles. Il produit une liste de décisions plus compacte (cf. annexe 2 et 3), mais est par conséquent également plus lent que d'autres algorithmes du même type.
PART génère par lui-même des listes de décision pour la classification. Plus précisément, il crée des arbres de décisions à chaque étape et suit la branche optimale.

Classification par Prix :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
ZeroR	?	?	0.431	<1s	?	?	0.476	<1s
JRip	0.826	0.826	0.826	2.01s	0.798	0.801	0.798	2.09s
PART	0.822	0.822	0.822	1.56s	0.839	0.843	0.839	1.58s

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 23  0  4 |  a = 0
  0 30  8 |  b = 1400
  5  3 51 |  c = 700
```

PART obtient, avec une courte avance les meilleurs résultats, avec un temps de traitement honorable. JRip suit de près, mais comme attendu, il est un peu plus lent. Cependant, comme dit plus haut, sa production de règles est plus lisible (cf. annexes 2 et 3)

Classification par Usage :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
ZeroR	?	?	0.531	<1s	?	?	0.526	<1s
JRip	0.974	0.974	0.974	0.56s	0.974	0.974	0.974	0.56s
PART	0.979	0.979	0.979	0.26s	0.983	0.983	0.983	0.26s

```
=== Confusion Matrix ===

  a  b   <-- classified as
 59  2 |  a = bureautique
  0 55 |  b = gaming
```

SMV :

L'algorithme Support Vector Machine travaille avec un sous-ensemble réduit de données, garde les solutions trouvées sur celui-ci, puis continue avec le reste des données encore non classées. Sequential Minimal Optimisation en est une amélioration, qui réduit le lourd coût en mémoire et le long temps de traitement d'un SVM classique.

Classification par prix :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
SMO	0.865	0.866	0.865	1.42s	0.839	0.850	0.839	0.064s

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 136   0  17 |   a = 0
   0 174  27 |   b = 1400
  24  16 228 |   c = 700
```

Comme prévu en amont, c'est ici la cross-validation qui prend le dessus.

Classification par Usage :

	Validation Croisée				Percentage Split			
	F-Mesure	Précision	Rappel	Tps de traitement	F-Mesure	Précision	Rappel	Tps de traitement
SMO	0.993	0.993	0.993	0.2s	1	1	1	0.19

```
=== Confusion Matrix ===

  a  b   <-- classified as
 61  0 |  a = bureautique
  0 55 |  b = gaming
```

Et ici, comme dans beaucoup de cas de classification par usage, c'est le percentage-split qui donne les meilleurs résultats (et quels résultats !).

Analyse des résultats

Les critères d'évaluation que nous avons utilisés sont le rappel, la précision, la F-mesure et le temps de calcul. Ces critères varient en fonction du classificateur et de la méthode d'entraînement du modèle – cross-validation et percentage split. Nous souhaitons à partir de ces critères déterminer la meilleur association modèle / méthode d'apprentissage pour chacune des classifications.

Concernant la classification des prix, globalement les meilleurs résultats en termes de précision, rappel et F-mesure se retrouvent lors de l'usage de la méthode de la cross-validation (10%). D'un point de vue de l'algorithme, les 2 meilleurs sont le LMT (Logistic Model Tree) de la famille des arbres de décision et le SMO de la famille des SMV. Sur les valeurs de précision, rappel et F-mesure, le LMT est légèrement meilleur que le SMO, toutefois le temps de traitement est très élevé – 32.39s pour LMT contre 1.42s pour SMO. Cela ne constitue pas un inconvénient majeur pour la taille de notre corpus, mais pourrait le devenir si nous recommencions ces tests avec un corpus de taille plus conséquente. C'est pour cela qu'en ce qui concerne la meilleur combinaison algorithme / méthode d'apprentissage pour la classification des prix d'ordinateur, notre choix se portera sur l'algorithme SMO avec une méthode de cross-validation (10%) qui est un très bon compromis entre performance et temps de traitement.

Du côté de la classification des usages, cette fois-ci, la méthode d'apprentissage qui engendre les meilleurs résultats est le percentage-split (80%-20%). Quant à l'algorithme avec les meilleurs performances, trois obtiennent un score parfait en termes de précision, rappel et F-mesure : LMT et Random Forest du côté des algorithmes de types arbres de précision, et SMO du côté des algorithmes de type SVM. Ces trois algorithmes se distinguent au niveau du temps de traitement : 0.1s pour LMT, 0.64s pour Random Forest, et 0.19s pour SMO. Pour la classification des usages des ordinateurs, la combinaison algorithme / méthode d'apprentissage avec les meilleurs résultats (et quels résultats !) est l'algorithme LMT avec un apprentissage en percentage-split (80%-20%).

Nous tenons tout de même à mentionner que les algorithmes JR48, JRip et PART bien que n'étant pas les algorithmes qui produisent les modèles les plus performants sont les modèles qui sont les plus lisibles et compréhensibles pour un humain. J48 nous a d'ailleurs aidé à vérifier nos données lors de l'application de la classification d'usage, dont les résultats étaient tellement beaux qu'ils en devenaient suspects. Nous avons cru avoir laissé passer un mot clé tel que « gaming » ou « bureautique » dans le corpus. Mais après visualisation de l'arbre de décision de JR48 et des règles construites par JRip et PART, nous avons pu constater que ce n'était pas le cas. Les algorithmes se basent vraiment sur des composants clés pour procéder à la classification par usage.

A la suite des résultats de la classification par usage nous nous sommes demandé la raison pour laquelle les critères semblaient si discriminant au niveau de l'usage et l'être moins dans la classification des prix (nous ferions-nous berner par certains vendeurs sur la qualité de l'ordinateur que nous achetons ?). La raison qui nous a paru la plus probable relèverait de la

construction de nos classification. En effet, la classification par usage est binaire, là où la classification par prix est ternaire, ce qui augmente le risque d'erreur. De plus, la classification des prix se faisant justement à partir des prix, certains ordinateurs doivent être à la limite de l'entrée de gamme et du milieu de gamme, et d'autres à la limite du milieu de gamme et du haut de gamme. Cette hypothèse s'est confirmée lorsque nous avons regardé plus attentivement les matrices de confusion.

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 136   0  17 |   a = 0
   0 174  27 |   b = 1400
  24  16 228 |   c = 700
```

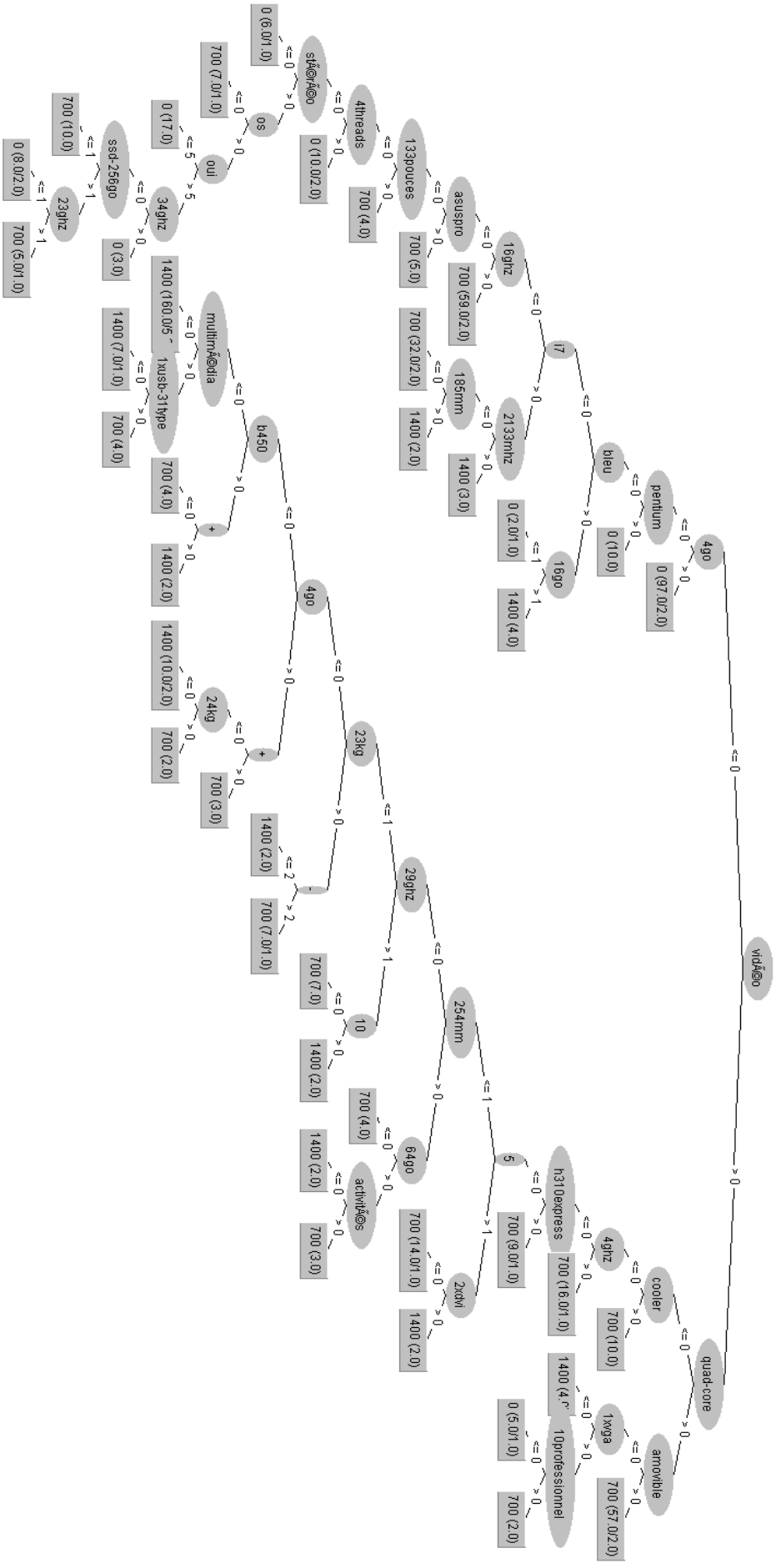
Comme le montre la matrice de confusion ci-dessus *(SMO en cross-validation)*, les erreurs de classification se situent entre les 0 (0-700€) et 700 (700-1400€), et entre 700 et 1400 (1400€ +). Jamais, ou alors très rarement, un élément de la classe 0 n'est classé dans la classe 1400, et inversement.

Conclusion

Nous avons conscience que, suivant le corpus et la classification effectuée, la combinaison entre l'algorithme et la méthode d'apprentissage la plus fructueuse est variable. Toutefois, bien que nous ayons conscience du caractère peu commun que revêtent nos données textuelles, pour une tâche de classification automatique nous conseillerions de tester les algorithmes SMO et LMT qui sont ceux avec lesquelles nous avons eux les meilleurs résultats lors de ce projet. Et en bonus nous préconiserions la mise en œuvre de l'algorithme JR48 qui permet d'avoir un aperçut des attributs les plus saillants du corpus et ainsi de s'assurer que les résultats ne soient pas biaisés par un attribut trop discriminant.

Il serait intéressant dans la poursuite de ce projet de faire varier le nombre de catégories d'une classification afin de déterminant s'il s'agit là d'un paramètre important dans le succès d'une tâche de classification automatique. Une suite logique de ce projet serait d'étudier la variation des paramètres des méthodes d'entrainements – cross-validation et percentage-split – ainsi que ceux des nombreux paramètres des algorithmes que nous avons utilisés.

Annexe 1 : arbre de décision de J48 pour les fourchettes de prix



Annexe 2 : ensemble de règles définies par JRip pour les fourchettes de prix

```
JRIP rules:
===========

(vidÃ©o <= 0) and (4go >= 1) => xClasse=0 (97.0/2.0)
(vidÃ©o <= 0) and (4mo >= 1) and (oui <= 5) => xClasse=0 (20.0/1.0)
(turbo <= 0) => xClasse=0 (23.0/7.0)
(12go >= 1) and (ssd-256go >= 2) => xClasse=0 (11.0/2.0)
(dual-core >= 1) and (0 >= 2) => xClasse=0 (5.0/1.0)
(geforce >= 1) and (non <= 9) and (6go <= 0) and (gtx-1050ti <= 0) => xClasse=1400 (110.0/6.0)
(12mo >= 1) and (oui >= 7) => xClasse=1400 (64.0/12.0)
(i7 >= 1) and (la <= 1) and (10famille <= 0) and (graphics <= 0) => xClasse=1400 (18.0/2.0)
 => xClasse=700 (274.0/36.0)

Number of Rules : 9
```

Annexe 3 : exemple de règles de PART (au nombre de 25 au total, liste complète dans le fichier « PART.txt »)

```
#######------------------------
règles PART :

PART decision list
------------------

vidéo > 0 AND
quad-core > 0 AND
amovible > 0: 700 (57.0/2.0)

vidéo > 0 AND
cooler <= 0 AND
n-300mbps <= 0 AND
4ghz <= 0 AND
h310express <= 0 AND
5 <= 1 AND
254mm <= 0 AND
23kg <= 0 AND
4go <= 0 AND
multimédia <= 0 AND
6xjack <= 0 AND
serial <= 0: 1400 (142.0/1.0)

i7 > 0 AND
2xmini <= 0 AND
3840x2160pixels <= 0 AND
850w <= 0 AND
bleu <= 0 AND
rtx-2060 > 0 AND
16go > 0: 1400 (10.0)

turbo <= 0 AND
i7 <= 0 AND
1go <= 0 AND
sata-6gbs <= 0: 0 (67.0)
```