Application Informatique Décisionnelle
Rapport

Fouille de données

Dembski Ludovic
Quillot Mathias

Master 2 ILSN
2016-2017

Table des matières

Table des matières **2**

Présentation **3**

Contexte 3

Organisation 3

Méthodes **4**

Données 4

Description du corpus 4

Prétraitement 5

Classificateurs 5

Modèles 5

Evaluation 6

Validation 6

Implémentation 6

Résultats **9**

Performance individuelle 9

Comparaison des outils de fouille 10

Performance générale 10

Conclusion **12**

Présentation

Contexte

Dans le cadre de l'UCE *Application spécialisée Aide à la décision* pour la sous-partie *Fouille de données* nous avons eu pour objectif de prédire les clients qui achèteront le livre "Histoire de l'art en Provence" à partir d'un jeu de données qui nous a été fourni.

Nous avons utilisé des modèles de *Multilayer Perceptron* dans la réalisation de cette tâche. Le but de ce document sera de vous présenter la méthode qui a été utilisée et les résultats qu'ont généré nos prédictions.

Organisation

Ludovic s'occupe des aspects fonctionnels liés à Python. Son objectif fut de coder les fonctionnalités en python et de faire des recherches concernant la manière d'utiliser les différentes classes de Weka via le système de commande.

Mathias s'est occupé d'une partie du code python, mais son objectif principal était de se documenter sur Weka et de proposer des modèles d'apprentissage sous accord commun avec Ludovic.

Dans une entente commune, une fois notre tâche finit, on expliquait chacun à l'autre ce que l'on avait appris et pourquoi nous avons eu ces résultats.

Méthodes

Données

Description du corpus

- **LocalId :** Numéro unique assigné au client dans l'échantillon.
- **GlobalId :** Numéro unique assigné au client dans la BD complète du libraire
- **Age :** âge du client, exprimé en années.
- **Gender :** (0 = femme, 1 = homme)
- **Region :** Région d'habitation du client (NW = nord-ouest, SW = sud-ouest, SE = sud-est, NE = nord-est)
- **CityCode :** Code de la ville du client (une lettre de A à S)
- **TotalSpent :** Somme totale dépensée chez le librairie.
- **TotalPur :** Nombre total de livres achetés.
- **FirstPur :** date du *premier* achat.
- **LastPur :** Nombre de mois depuis le premier achat.
- **NovelPur :** Nombre de romans achetés.
- **ChildPur :** Nombre de livres pour enfants achetés.
- **YouthPur :** Nombre de livres pour adolescents achetés.
- **CookPur :** Nombre de livres de cuisine achetés.
- **DiyPur :** Nombre de livres de bricolage achetés.
- **RefPur :** Nombre d'encyclopédies et de dictionnaires achetés.
- **ArtPur :** Nombre de livres d'art achetés.
- **GeogPur :** Nombre de livres de géographie achetés.
- **Provcook :** Nombre d'achats du livre "Les secrets de la cuisine provençale".
- **ProvAtlas :** Nombre d'achats du livre "Atlas historique de la Provence".
- **ProvArt :** Nombre d'achats du livre "Tout sur l'art provençal".
- **RelPur :** *Classes cibles*, indique si le client a acheté "Histoire de l'art en Provence" (Yes) ou pas (No).

Attention : Dans cette base de données, certaines valeurs sont notées NA : il s'agit de valeurs inconnues.

La suite de cette partie décrira les différents prétraitement que l'on aura réalisé et qui seront utilisés pour les différents modèles.

Nous avons des individus **ne possédant pas de valeur** pour certains de leurs champs. Voici la liste des champs et le nombre d'individus n'ayant pas de valeur pour celui-ci :

- Age 173
- CityCode 97
- Les autres champs 0

Prétraitement

Nous nous sommes focalisés de différents modèles, c'est pourquoi nous n'avons réalisé qu'un prétraitement du corpus initiale qui sera utilisé par tous les modèles.

La première étape de notre prétraitement consiste à transformer le corpus, notamment en le nettoyant, réalisant un recodage, en transformer les dates, ou encore en le convertissant de CSV vers ARFF. Ces tâches sont exécutés dans l'ordre suivant :

- **Done Nettoyage** Suppression des individus avec valeurs manquantes. (ex: age)
- **Done numérisation** transformation des dates en timestamp
- **Done Recodage** Disjonction sur Region, CityCode et Gender
    - weka.filters.supervised.attribute.NominalToBinary
- **Done Prétraitement** Conversion CSV vers ARFF
    - weka.core.converters.CSVLoader

L'autre étape du prétraitement consiste à découper le corpus en trois parts : **train**, **dev**, **test**. Nous avons décidé de maintenir la même distribution des classes dans chacun des jeux de données.

Classificateurs

Modèles

Nous nous sommes principalement concentré sur l'algorithme *Multilayer Perceptron*. Il permet de créer des modèles à différentes couches de perceptrons avec des fonctions d'activation.

A savoir que nous avons utilisé d'autres méthodes de Bayes mais qu'elles ne donnaient pas résultats escomptés. En effet, elle prédisaient tous les individus à la classe *no* minimisant ainsi le nombre d'erreur. Ceci est dû au fait que les deux classes du corpus sont déséquilibrées. D'autres approches auraient pu être essayées comme l'équilibrage des données en utilisant autant d'individus de classes *yes* que de classe *no* ou encore en donnant utilisant le *cost sensitive* pour plus de poids à la classe *yes* mais par un soucis de temps et une volonté de tester le *Multilayer Perceptron* nous avons préféré ne pas nous concentrer dessus.

Notre idée fut de tester différentes valeurs des trois paramètres :

- **Learning Rate (option L) :** cette valeur est un facteur associé à la fonction de propagation des erreurs sur l'apprentissage. Elle a un impacte direct sur le résultat.
- **Momentum (option M) :** valeur qui influence aussi la propagation des erreurs.

- **Number of Epochs (option N) :** nombre d'itérations réalisées. Valeur qui influence sur le temps de calcul. Pour la valeur 10, l'apprentissage se calcule instantanément contrairement à la valeur 500 qui prend quelques minutes.

Nous avons configuré le script pour que les valeurs par défaut soient les suivantes : *L* (0.3), *M*(0.3), *N*(20).

Nous avons fait varier chaque valeur pour tester différentes configurations de l'algorithme. Les autres valeurs utilisées étaient celles par défaut. La valeur *L* a varié de 0,1 à 1 avec un pas de 0,1, la valeur M a varié de 0,1 à 1 avec un pas de 0,1 et la valeur N a varié de 10 à 460 avec un pas de 50.

Evaluation

Validation

La validation se fait en trois étapes dans notre script. La première fut d'exécuter la prédiction sur chaque modèle avec le jeu de données *dev*. La deuxième étape fut celle de trouver pour chaque modèle le seuil de séparation avec la meilleur f-measure pour la classe yes. Enfin, la troisième étape fut de comparer les valeurs *f-measure* de chaque modèle et de choisir le meilleur, en gardant en mémoire son seuil pour le réutiliser lors des tests.

Une question que l'on s'est posé est *"comment mesurer la performance du modèle ? Précision ? Rappel ? F-measure ? D'autres idées ? Sur quel variable ? Les deux ? "*. Avant de répondre à la question, il a fallu bien comprendre le problème. L'utilisateur veut prédire un maximum d'acheteur du livre parmis ceux qui l'achèteraient. Il faut donc maximiser le rappel. Cependant, il faut éviter les erreurs car il souhaite, via ces résultats, prédire les retombées financières des achats. Il faut donc aussi maximiser la précision. L'idée est donc d'utiliser la **f-measure** de la **classe *yes*** afin de trouver un équilibre entre les deux mesures de performance.

Test

Le test consiste à prédire les classes *yes* et *no* des individus en se basant sur le jeu de données *test* et en utilisant le seuil trouvé lors de l'étape de validation. Ainsi, nous avons récupéré la f-measure de notre modèle et son taux d'erreur.

Implémentation

Nous avons décidé d'utiliser le python pour exécuter les différentes tâches d'apprentissage.

Corpus Pour réaliser cette tâche, nous avons donc utilisé la fonction weka suivante : **weka.filters.supervised.instance.Resample**.

Pour utiliser le script, il suffit d'exécuter python avec le fichier du script en paramètre, celui-ci se nommant *ScriptFinal.py*. Exemple : *python ScriptFinal.py*.

Attention ! Ce script doit être exécuté **avec python 2** ! Python3 ne pourra pas l'exécuter.

Le script exécute les tâches suivant l'ordre de la liste ci-dessous :

1. **Prétraitrement**
    a. Changement de séparateur CSV ";" vers ","
    b. Suppression des instances ayant des valeurs non définies (NA)
    c. Transformation des dates en timestamp
    d. Transformation du fichier en arff : weka.core.converters.CSVLoader
    e. Disjonction sur les variables nominales (sauf la classe) : weka.filters.supervised.attribute.NominalToBinary
    f. Découpage du corpus en train dev test : weka.filters.supervised.instance.Resample
2. **Creation des modèles**
    a. Modèles MultilayerPerceptron avec différents paramètres : weka.classifiers.functions.MultilayerPerceptron
3. **Validation**
    a. Calcul des meilleurs f-measure sur la classe *yes* de chaque modèle en gardant en mémoire le threshold associé.
    b. Sélection du meilleur modèle. - avec la meilleur f-measure.
4. **Test**
    a. Test du modèle sélectionné dans l'étape de validation et affichage de sa f-measure et de son error rate.

Information concernant la classe Resample de Weka. Si on lance le filtre *Resample* deux fois, il risque de donner deux découpes différentes du corpus - caractéristique non déterministe - c'est pourquoi le script ne réalise plus le prétraitement une fois qu'il détecte que les fichiers train, dev et test existent déjà, permettant de garder à chaque exécution le même jeu de données lorsque l'on souhaite calculer de nouveaux modèles. Il suffit de supprimer les fichiers et de relancer le script pour que le corpus soit à nouveau calculé.

Le processus de prétraitement génère les fichiers train, dev et test dans le dossier data ayant les noms suivant :

- KTFGHU14_train
- KTFGHU14_dev
- KTFGHU14_test

Les fichiers de sorties du calcul des modèles sont mis dans *models/model2* et sont les suivants :

- ***[prefix]-modelTrained.model*** **:** Fichier du model qu'on pourra ensuite charger pour les validations et les tests
- ***[prefix]-trainOutput.txt*** **:** Sortie console de l'apprentissage.
- Prefix étant la configuration du modèle. (ex: L_0.2)

Au niveau du script, la validation a généré dans le dossier models/model2 les fichiers suivants :

- **[prefix]-classificationOutput.csv :** prédiction de chaque instance (individu).
- **[prefix]-thresholdOutput.arff :** weka en exécutant l'algorithme sur les données *dev* a calculé toutes les possibilités de changement de classe des individus. Il fait apparaitre dans ce fichier chaque changement de classe d'un individu avec les métriques associées et le threshold (seuil) qui permet ce changement. Il est, logiquement, trié par threshold croissant.
- **[prefix]-thresholdOutput.csv :** version csv de thresholdOutput.arff.
- Prefix étant la configuration du modèle. (ex: L_0.2)

Voici la liste des fichiers généré dans le dossier models/model2 après avoir fait les tests :

- **[prefix]-classificationOutput.csv :** même que pour la validation mais sur le jeu de données test.
- **[prefix]-thresholdOutput.arff :** même que pour la validation mais sur le jeu de données test.
- **[prefix]-thresholdOutput.csv :** version csv de thresholdOutput.arff.
- Prefix "test-yes" ou "test-no" selon la classe qui a été prise en considération pour faire les calculs. (en effet, on génère les deux pour avoir plus d'informations sur nos résultats)

Le code est accessible sur github : https://github.com/dem-l/LeLibraire.git

Résultats

Performance individuelle

Voici les performances de chaque modèle généré (Multilayer Perceptron) pendant la phase de validation.

Variable	Valeur	F-measure	Threshold
L	0.1	0.196078	0.119159
L	0.2	0.198347	0.084317
L	0.3	0.203279	0.065664
L	0.4	0.197452	0.059984
L	0.5	0.188088	0.05527
L	0.6	0.191781	0.060117
L	0.7	0.188312	0.061712
L	0.8	0.185792	0.048138
L	0.9	0.192308	0.047347
L	1	0.186916	0.049245
M	0.1	0.200873	0.0857887
M	0.2	0.201754	0.085644
M	0.3	0.203279	0.065664
M	0.4	0.2	0.062872
M	0.5	0.192547	0.057183
M	0.6	0.188679	0.054093
M	0.7	0.191419	0.056235
M	0.8	0.176373	0.010868
M	0.9	0.173554	0.028325
M	1	0.155556	0
N	10	0.192308	0.074606

N	60	0.173134	0.065681
N	110	0.178182	0.011355
N	160	0.173516	0.02135
N	210	0.163978	0.000008
N	260	0.168421	0.000732
N	310	0.169279	0.001643
N	360	0.170068	0.000001
N	410	0.172932	0.000004
N	460	0.171821	0.000001

Le modèle qui a été retenu pendant la phase de validation est celui avec la valeur L(0.3), M(0.3) et N(20) avec une f-measure de 0.203279 et un seuil qui vaut 0.065664

Ce que l'on peut observer est que L et M n'ont pas une grande incidence sur le résultat bien que L à (0.3) et M(0.3) donnent les meilleurs résultats.

Autre chose à constater, le pas choisi sur N est certainement trop élevé. En effet, il serait intéressant de voir entre 10 et 60 si on peut observer un maximum local que l'on n'a pas pu faire ressortir dans notre expérience.

A savoir que ce paramètre N devrait être choisi dans l'étape d'apprentissage. L'algorithme se serait servi du jeu de données de *dev* durant l'apprentissage pour éviter le sur-apprentissage en vérifiant les performances du dev à chaque itération et en s'arrêtant à la valeur de N la plus performante.

Comparaison des outils de fouille

Aucune comparaison n'est réalisable dans notre cas puisque l'on n'a utilisé que le Multilayer Perceptron.

Performance générale

Nous l'avons ensuite testé sur le jeu de données de *test*. Il faut savoir que pendant cette phase, le seuil choisi peut différer de celui de la validation - n'impliquant pas de changement sur le résultat. En effet, nous le recherchons dans le fichier thresholdOutput généré et celui-ci peut ne pas exister dedans. Nous prenons alors le seuil inférieur le plus proche puisque la différence entre les deux seuils n'induit pas de changement de classe pour des individus.

Nous avons finalement réalisé les tests sur le modèle choisi précédemment. Les résultats de ce test sont les suivants :

- F-measure sur la classe Yes de 0.202091
- F-measure sur la classe No de 0.812552 (à titre d'information supplémentaire)

- Un taux d’erreur de 30.6%

Conclusion

Au cours de ce projet, nous avons été mis en situation et confronté aux problématiques de la fouille de données. Comment à partir d'un corpus arriver à un modèle de prédiction qui soit performant. Comment découper son jeu de donnée ? Comment organiser la phase de train, dev, test ? Quel mesure de performance utiliser pour choisir le modèle qui correspond le mieux à notre problème ?

Nous avons trouvé que le sujet était intéressant, le jeu de données à la fois simple et complet puisqu'il était facile à comprendre et qu'il nous a soumis à des réflexion concernant les données manquantes, le problème d'équilibrage des classes ou encore la transformation de certaines variables - notamment les variables nominales ou la date.

Un problème qui fut difficile à surmonter fut de se documenter sur Weka. Il serait peut être utile de nous donner une liste de classes qui pourraient nous intéresser avec les options de sorties pour que l'on puisse récupérer les données intéressantes. En effet, la documentation de Weka n'est pas très clair concernant les sorties des prédictions ou des seuils de décision.

Le projet est assez chronophage. On peut vite tomber dans des problèmes difficilement solvables sur lesquels il faut passer du temps à chercher dans la documentation et sur les forums sur le Web. Le projet a peut être commencé un peu tard : comparé au premier projet de la matière, il s'est étalé sur moins de temps et a demandé beaucoup plus d'énergie.

Nous avons utilisé différents modèles avec différents algorithmes d'apprentissage. La plupart de ces modèles prédisent tous les individus dans la même classe. Lorsque nous avons utilisé l'algorithme MultiLayerPerceptron, nous avons eu des résultats différents, c'est pourquoi nous nous sommes focalisés dessus. Le problème est que nous n'avons peut-être pas dépenser notre énergie dans les bonnes tâches et nous n'avons peut-être pas pris les bonnes décisions ce qui ne nous a pas permis de scripter des modèles avec d'autres méthodes pour les comparer par la suite.

C'est d'ailleurs une des propositions que nous faisons : passer à d'autres algorithmes d'apprentissage et de les tester. Nous proposons aussi d'utiliser le cost sensitive afin de donner plus de poids à la classe *yes*. Une autre proposition serait de créer d'autres prétraitement des données en supprimant par exemple les classes ayant des valeurs manquantes au lieu de supprimer les individus. Nous pourrions aussi garder autant d'individus de classe *no* que de classe *yes* pour équilibrer le jeu de données, le problème se porterait alors sur "quels individus garder "(grâce à des statistiques) et "est-ce que le jeu de données ne serait pas trop faible ?".