Introduction à l'apprentissage machine (GIF-4101 / GIF-7005)
Département de génie électrique et de génie informatique
Automne 2018

UNIVERSITÉ LAVAL

Devoir 5

Instructions :

- Formation des équipes :
  - GIF-4101 : le devoir est réalisé en équipe de deux à trois étudiants
  - GIF-7005 : le devoir est réalisé individuellement
- Programmation :
  - Produisez vos solutions dans les fichiers fournis, en respectant les instructions et n'utilisant pas d'autres modules que ceux qui y sont importés
  - Les temps de calcul sont vérifiés dans le code. Toute mention d'une durée trop importante entraînera *la note de zéro (0)* pour la sous-question correspondante
- Remise :
  - La remise du rapport et du code source se fait dans monPortail
  - Le remise doit être effectuée au plus tard le mercredi 12 décembre à 9h30
- Pondération :
  - Ce devoir compte pour 5% de la note finale

---

1. **Clustering de données (5pt)**

   Soit le jeu de données *Wisconsin Breast Cancer*, disponible dans scikit-learn [1]. Les 30 variables correspondent à différentes mesures effectuées sur des tumeurs devant être classifiées comme bénignes ou malignes. Partez du fichier `d5q1.py` [2] pour implémenter votre solution.

   (a) En utilisant l'algorithme K-means, faites varier le nombre de clusters utilisés selon $K = 2,4,\ldots,50$ et comparez les résultats en utilisant les mesures de l'indice de Rand ajusté [3], le score basé sur l'information mutuelle [4] et la mesure V [5]. Pour déterminer l'étiquette prédite, assignez au cluster l'étiquette (tumeur ou non) la plus fréquente parmi les données appartenant au cluster. Tracez les résultats dans une figure comparant la valeur des mesures de performance selon le nombre de clusters. Indiquez quel serait d'après vous le bon nombre de clusters à utiliser.

   (b) Répétez la question 1(a) précédente, avec $K = 2,4,\ldots,50$, en utilisant cette fois l'implémentation de l'algorithme EM pour une loi normale, initialisé aléatoirement [6].

   Pour évaluer les performances du clustering de EM, il faut assigner à chaque donnée une probabilité a posteriori de classement $P(C_i|\mathbf{x}^t)$ selon :

$$P(C_i|\mathbf{x}^t) = \frac{\sum_j P(C_i|\mathcal{G}_j)\, P(\mathcal{G}_j|\mathbf{x}^t)}{\sum_i \sum_j P(C_i|\mathcal{G}_j)\, P(\mathcal{G}_j|\mathbf{x}^t)}, \qquad P(C_i|\mathcal{G}_j) = \frac{\sum_t r_i^t P(\mathcal{G}_j|\mathbf{x}^t)}{\sum_i \sum_t r_i^t P(\mathcal{G}_j|\mathbf{x}^t)}.$$

1. `datasets.load_breast_cancer`
2. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d5q1.py`
3. `metrics.adjusted_rand_score`
4. `metrics.adjusted_mutual_info_score`
5. `metrics.v_measure_score`
6. `mixture.GaussianMixture(init_params='random')`

Le classement se fait en assignant la donnée à la classe dont la probabilité de classement est la plus élevée :

$$y_i^t = \begin{cases} 1 & \text{si } i = \text{argmax}_l \, P(C_l|\mathbf{x}^t) \\ 0 & \text{autrement} \end{cases} .$$

(c) Répétez la question 2(b), mais en utilisant initialisant cette fois l'algorithme EM à partir d'un clustering effectué par K-Means. Pour le reste, utilisez la même procédure et rapportez les résultats pour $K = 2,4,\ldots,50$ clusters.

(d) Analysez les résultats de K-means, EM avec initialisation aléatoire et EM avec initialisation utilisant K-means. Assurez-vous de répondre aux questions suivantes dans votre analyse.
— Pourquoi EM performe-t-il significativement mieux lorsque ses clusters sont initialisés en utilisant K-means ?
— Qu'est-ce qui explique le pic de performance à $K = 2$ pour EM initialisé avec K-means ?
— Si l'on compare K-means et l'algorithme EM pour une loi normale multivariée, donnez les principales ressemblances et différences que l'on peut noter.

2. **Sélection de variables (5pt)**

Soit le jeu de données *CSDMC2010 SPAM corpus*, dont nous avons extrait un jeu de données avec 1000 caractéristiques binaires, indiquant la présence ou non d'un mot particulier. Ce jeu binarisé est disponible sur le site Web du cours [7]. L'archive zip comporte trois fichiers :
— `data` : les données, chaque ligne étant une instance et chaque colonne une caractéristique ;
— `target` : les étiquettes de classe des données, 0 indiquant que l'instance est un pourriel, 1 indiquant que c'est un message légitime ;
— `feature` : la liste des 1000 mots d'intérêt, dans le même ordre que les colonnes de `data`.
Supposons que l'on veut sélectionner les variables d'intérêt de ce jeu de données, où chaque variable indique la présence ou non d'un mot particulier. Pour cette question, utilisez le fichier `d5q2.py` [8] pour votre implémentation et servez vous de la partition entraînement/test déjà effectuée dans le code.

(a) Effectuez une sélection univariée en conservant les 10 meilleures variables parmi les 1000, selon le test [9] du χ^2 et le critère d'information mutuelle [10]. Effectuez cette sélection sur la partition d'entraînement. Rapportez les variables (et les mots correspondants) sélectionnées par les deux critères de sélection utilisés. Rapportez également la performance de classement sur la partition de test, en utilisant un SVM linéaire [11] comme modèle de classement généré sur la partition d'entraînement. Comparez les résultats avec ceux obtenus avec la même configuration, mais utilisant l'ensemble des 1000 variables.

(b) Répétez les expérimentations de la sous-question précédente avec la sélection séquentielle arrière implémentée dans la fonction `feature_selection.RFE` de scikit-learn. Faites les expérimentations en utilisant un SVM linéaire comme modèle de base, en faisant un réentraînement à chaque itération de l'algorithme (`step`=1).

(c) Expliquez pourquoi un algorithme de sélection arrière séquentielle est souvent préféré à un algorithme de sélection avant séquentielle pour des données comportant des dépendances non linéaires complexes entre les variables. Expliquez également pourquoi l'algorithme de sélection séquentielle arrière peut difficilement procéder à un nouvel entraînement pour chaque

7. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/csdmc-spam-binary.zip`
8. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d5q2.py`
9. `feature_selection.chi2`
10. `feature_selection.mutual_info_classif`
11. `svm.LinearSVC`

sous-ensemble candidat de variables pour des cas où il y a un grand nombre de variables (D grand) et un nombre relativement faible de variables à sélectionner ($K \ll D$), comme c'est le cas pour une sélection de 10 variables parmi 1000.

3. **Méthodes par ensemble**

   Le fichier `d5q3.py` [12] contient du code générant le jeu de données suivant :

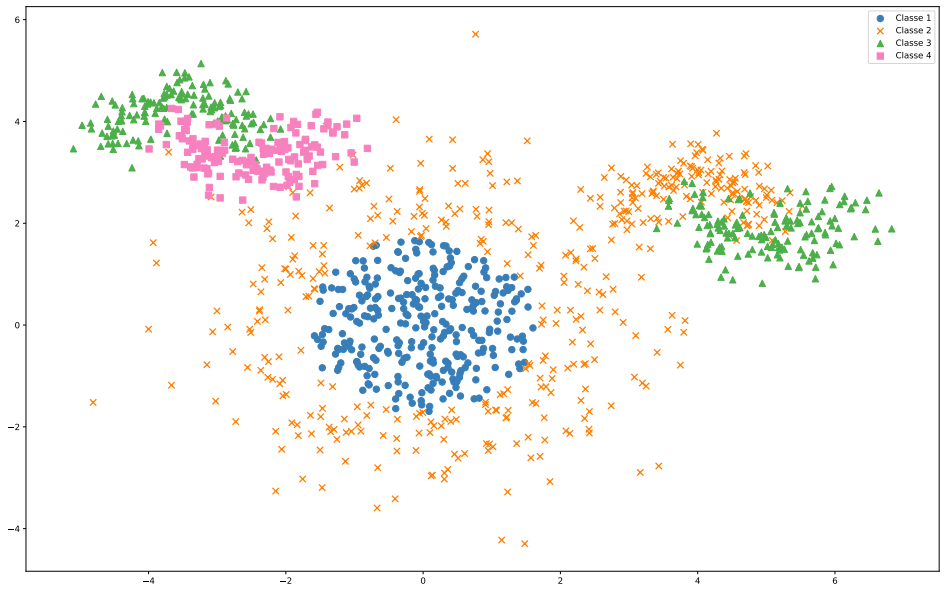

FIGURE 1 – Jeu de données à 4 classes généré par `d5q3.py`.

Utilisez ce jeu pour répondre aux questions suivantes. Ne le modifiez pas et ne changez pas la division entraînement/validation telle qu'effectuée dans le code.

(a) Utilisez l'implémentation de scikit-learn pour évaluer la performance d'un classifieur par ensemble basé sur *AdaBoost*, en utilisant comme classifieurs de base (*weak learners*) :
   — Des souches de décisions (arbres de décision à 1 niveau) [13] ;
   — Des arbres de décisions à 3 niveaux [14].

   Dans les deux cas, utilisez un maximum de 50 classifieurs de base (`n_estimators=50`).

   Pour ces deux configurations, rapportez les résultats à l'aide des figures suivantes :
   — Un graphique de la performance en entraînement et test en fonction du nombre de classifieurs de base utilisés (de 1 à 50) ;
   — **Trois** figures montrant les régions de décision de l'ensemble pour différentes valeurs représentatives du nombre de classifieurs de base et les points du jeu de données. Assurez-vous d'utiliser des couleurs similaires pour les régions et les marqueurs.

   Note : les fonctions `staged_predict` et `staged_score` fournies par AdaBoostClassifier pourraient vous être très utiles ici pour réduire l'effort de calcul requis.

(b) Discutez brièvement des résultats obtenus. Dans votre discussion, assurez-vous de répondre aux questions suivantes :
   — Des souches de décisions sont-elles suffisantes pour ce genre de problème ?

12. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d5q3.py`
13. `DecisionTreeClassifier(max_depth=1)`
14. `DecisionTreeClassifier(max_depth=3)`

— Quel serait approximativement le nombre de classifieurs optimal pour ce jeu de données ?
— Pourquoi existe-t-il, à la fin de l'apprentissage, des zones de l'espace de recherche ne comportant *que* des exemples d'une classe en entraînement, mais faisant néanmoins partie des régions de décision d'autres classes ?

(c) Utilisez maintenant une *forêt aléatoire* (*Random Forest*) avec 50 classifieurs de base [15]. Faites varier la profondeur maximale des arbres de décision de 1 à 12 inclusivement. Produisez une figure montrant la performance en entraînement et test en fonction de la profondeur maximale utilisée, ainsi que quatre figures des régions de décision pour des profondeurs de $\{1, 4, 8, 12\}$. Quelle semble être la profondeur optimale pour ce problème ?

4. **Réduction de dimensionnalité**

Utilisez le jeu de données *digits* [16] pour comparer différentes méthodes de réduction de dimensionnalité. Pour cette question, vous devez comparer trois méthodes :
— Analyse en composantes principales (PCA) [17] ;
— Positionnement multidimensionnel (*Multidimensional scaling, MDS*) [18] ; dans le cas de cette méthode, utilisez `n_init=1` pour minimiser le temps de calcul ;
— t-SNE [19].

Appliquez chacune de ces méthodes au jeu de données *digits* pour réduire sa dimensionnalité de 64 à 2, de façon à pouvoir l'afficher dans une figure. Une fonction nommée `plot_clustering` vous est fournie pour la création de la figure dans le fichier `d5q4.py` [20].

Produisez les résultats suivants pour chaque méthode et discutez brièvement de ceux-ci.

(a) La visualisation 2D telle que produite par `plot_clustering`.

(b) Des images correspondant à quelques exemples de données appartenant à des classes différentes mais néanmoins placées à proximité dans l'espace 2D (les données de *digits* sont des images de 8x8 pixels, que vous pouvez donc afficher).

(c) Pour chaque méthode et chaque classe du jeu de données, le ratio entre la distance euclidienne *intra-classe* (la distance moyenne entre tous les exemples d'une même classe, une fois la méthode de réduction de dimensionnalité appliquée) et la distance *inter-classe* (la distance moyenne entre les exemples d'une classe et tous les autres exemples n'étant pas de cette classe). Pour cette sous-question, produisez également les résultats pour le jeu de données original à 64 dimensions. La fonction `cdist`, fournie par *scipy*, pourrait vous être utile ici.

Note : la durée d'exécution n'est pas mesurée pour cette question puisque tous les algorithmes utilisés sont directement ceux de scikit-learn. Assurez-vous toutefois que votre script s'exécute en un temps raisonnable (moins de 3 minutes).

28/11/2018 (rév. : 28/11/2018)
CG & MAG

15. `RandomForestClassifier(n_estimators=50)`
16. `datasets.load_digits()`
17. `decomposition.PCA`
18. `manifold.MDS`
19. `manifold.TSNE`
20. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d5q4.py`