Projets finaux d'IA

Projet 2: Régression Logistique / Naive Bayes

À rendre avant le *13 Janvier 2024*

1 Description

Le but de ce projet, qui prolonge le TP5, est l'étude approfondie des technique de Régression Logistique et Naive Bayes comme outils d'apprentissage supervisé.

2 Questions

- Réalisez les tâches ci-dessous en proposant et documentant vos propres implémentations. Le détail technique peut être trouvé dans le cours et dans les livres proposés dans le cours [1, 11, 4, 2, 9, 3, 7, 6]. L'idée générale est d'utiliser la librairie Python SciKitLearn (SKLearn, ou une autre de votre choix) comme référence.
- Comparez systématiquement dès que cela est possible vos résultats au classifieur de référence k-NN (voir Annexe), avec $k = 1$ par défaut, ou k issu d'un entraînement.

  Il est a noter que rien ne vous empêche (pour pratiquer) de comparer vos développements avec d'autres techniques de SKLearn résolvant la même tâche
- Pour chaque tâche assurez vous de toujours bien documenter quel langage a été utilisé, quelle librairie éventuelle, quelles données, quels paramètres, ... Si vous avez lu ou utilisé de la documentation externe, assurez-vous de bien en citer la référence.
- Pour rendre chaque résultat obtenu et présenté utile, proposez un commentaire et une interprétation de ce résultat ("ce résultat illustre bien que...", "ce résultat confirme les limites du modèle...", ...)
- Rassemblez votre étude et résultats dans un document PDF à la manière d'un article scientifique [5, 8, 10, 12] qui pourra être généré soit (de préférence) par LaTeX (éventuellement en utilisant OverLeaf ou LyX) ou avec Word ou OpenOffice.

Note: Quand il est demandé de donner "x en fonction de y", on attend un graphique avec en abscisse x et en ordonnée y, avec éventuellement la variance (par exemple, box plot) sur chaque point.

3 Méthode

Sur la base d'ensembles de données le projet vise à:

1. Implémenter la méthode de Descente en Gradient. Tester sur la fonction $y = x\cos(\pi(x+1))$ en caractérisant les minima locaux (on pourra aussi commencer le test sur une fonction quadratique).
2. Implémenter l'algorithme de Régression Logistique
3. Implémenter la méthode de Naive Bayes

4. Pour la suite comparer systématiquement les 2 techniques:
   (a) Implémenter les mesures d'évaluation Précision, Rappel, F1 score, Accuracy
   (b) Donner l'évolution de ces mesures en fonction du volume de données utilisées pour l'apprentissage
   (c) Donner l'évolution de ces mesures en fonction du niveau de bruit dans les données d'apprentissage
   (d) Donner l'évolution de ces mesures en fonction du niveau de bruit dans les données de test
   (e) En utilisant un volume de données d'apprentissage réduit, mettre en évidence les phénomènes de sur-apprentissage
5. Une fois les paramètres des classes obtenus en supposant l'indépendance conditionnelle de caractéristiques Gaussiennes, commenter la structure des données (séparabilité, proximité des classes, similarité de leurs structures,...).
6. Pour boucler le circuit des données, en utilisant ces mêmes paramètres, échantillonner de nouvelles données dans chaque classe pour lesquelles on confirmera la vraisemblance et la classification.
7. Mettez vos resultats en parallèle avec ceux obtenus par la librairie SKLearn: sklearn.linear_model.LogisticRe et sklearn.naive_bayes.GaussianNB

4 Données

Le site UCI Machine Learning repository propose un grand nombre d'ensembles de données "standards" sur des tâches incluant classification et régression. On se propose de se concentrer en priorité sur les ensembles:

- *Mushroom* pour les données catégorielles
- *Wine quality* pour des données réelles

Ces ensembles ont suffisamment de données pour être facilement manipulés et pour être représentatifs. On pourra sous échantillonner ces données pour le développement. On pourra aussi appliquer le protocole classique vu en cours avec *hold-out* et validation croisée.

A noter que ce site contient aussi des ensembles très classiques comme *Iris* ou *Breast Cancer*.

Les sites proposant des données incluent le site de défis IA Kaggle et bien d'autres.

L'ensemble de données MNIST est aussi un ensemble de données qui a fait référence dans le développement des méthodes d'IA. Sa classification étant maintenant considérée comme "trop simple" , il a été dérivé dans des ensembles annexes du type *fashion MNIST*.

Au delà des ensembles *Mushroom* et Wine quality, on pourra utiliser d'autres ensembles (y compris générés artificiellement) pour le développement et le test avec la condition de les documenter correctement.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[5] Yacine Izza, Alexey Ignatiev, and João Marques-Silva. On explaining decision trees. *CoRR*, abs/2010.11034, 2020.

[6] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[7] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press, 2003. (available online).

[8] B. Mehlig. Artificial neural networks. *CoRR*, abs/1901.05639, 2019.

[9] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass., 2013.

[10] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[11] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[12] Jinxiong Zhang. Dive into decision trees and forests: A theoretical demonstration. *CoRR*, abs/2101.08656, 2021.

Annexe: Classifieur k-NN

Le classifieur k-NN (k-*Nearest Neighbor classifier*) permet de résoudre une tâche de classification potentiellement sans entraînement. Il fonctionne en associant à la donnée q (*query*) que l'on cherche à classifier le label de classe majoritaire parmi les k plus proches voisins de q.

Ce classifieur fait donc l'hypothèse d'une cohérence du label de classe autour de chaque point donné. Il fait donc aussi l'hypothèse de l'existence d'une distance (par exemple Euclidienne) qui permet de définir les termes “voisins” et “autour” ci-dessus.

Le seul paramètre de ce classifieur est le paramètre k réglant la taille du voisinage considéré. Un autre degré de liberté est la façon dont on définit "majoritaire". On peut simplement prendre le label le plus fréquent, ou le label le plus présent au sens de la proximité de ses points de support (on pondère chaque voisin par l'inverse de sa distance à q).

On notera que, au moment de la classification d'une donnée de label inconnu, ce classifieur fait appel au k plus proches voisins dans l'ensemble de référence (d'entraînement) de la donnée à classifier. Ceci n'est pas le cas dans le cas général, où les données de référence sont résumées dans (représentées par) la structure apprise (arbre de décision, réseau de neurones, régression,...). Ceci pose donc le problème d'identifier les k plus proches voisins, qui est un problème classique

d'indexation (*k-Nearest Neighbor search*) pour lesquelles des solutions plus ou moins optimales, comme le *k-d-tree*, existent.

4.1 Entraînement

On peut régler k en le choisissant par entraînement. On choisit la valeur, selon un intervalle $\{0, \cdots, k_{\max}\}$, qui réduit la valeur de l'erreur d'entraînement. Dans ce cas, l'erreur de test sert à valider la généralisation de cet entraînement. On peut bien sûr utiliser la validation croisée.

4.2 Fonction de distance

Quand les données sont réelles ($\mathbb{R}^D$) ou entières ($\mathbb{Z}^D$), la fonction de distance à utiliser la plus naturelle est la distance Euclidienne (distance de Minkowski pour $p = 2$ ou norme L_2) ou la distance de Manhattan (distance de Minkowski pour $p = 1$ ou norme L_1).

Si les données sont catégorielles (par exemple Température $T \in \{\text{chaud}, \text{froid}\}$), on peut coder les données sous forme de *one hot vector*:

1. On crée un vecteur binaire dont la longueur l est le nombre de catégories. Par exemple $l = 2$ pour $T \in \{\text{chaud}, \text{froid}\}$.

2. On "encode" chaque catégorie en mettant à '1' le bit correspondant à la catégorie. Dans l'exemple, "chaud" sera représenté par 10 et "froid" par 01. On note que l'on peut aussi encoder des données appartenant à plusieurs catégories en mettant à '1' les bits correspondants aux catégories concernées.

On peut alors utiliser la distance de Hamming sur ces représentations binaires. La distance de Hamming est basée sur le nombre de bits à '1' en commun dans les 2 représentations à comparer. Si chacune des données n'appartient qu'à une et une seule catégorie, alors la distance sera soit 0 soit 1.

4.3 Implémentation

Le classifieur k-NN est proposé dans la librairie SKLearn sous l'interface sklearn.neighbors.KNeighborsClassifier.

Vue la simplicité de son implémentation, il est recommandé d'essayer sa propre implémentation en créant la même interface pour tous les classifieurs afin qu'ils soient interchangeables dans les tests de performance.