TP 10 – Machine learning avec Python et Scikit-Learn
Année 2022-2023

L'objectif de ce TP est d'aborder l'apprentissage automatique supervisé en déroulant toute la méthodologie algorithmique associée : chargement des données, séparation données d'entraînement et de test, application de deux méthodes d'apprentissage vues en cours (régression logistique et k-NN) pour prédire la classe de données de test, évaluation de chacune des méthodes.

Vous vous appuierez sur la librairie Python `Scikit-learn`. Vous travaillerez sur le jeu de données de chiffres `MNIST` composé d'images de chiffres manuscrits à une résolution de 8*8. En `Scikit-learn`, elles se nomment `digits`.
Le code vous est fourni. Il vous est demandé de bien comprendre toutes les lignes de code en Python, de lire la documentation associée à chacune des fonctions/méthodes utilisées, et de manipuler leurs paramètres afin de comprendre leur utilité. Vous commenterez ces lignes de code afin de vous les approprier.

Machine learning pour les données *digits* (`MNIST`)

Dans cette partie vous allez travailler sur des données de chiffres manuscrits disponibles dans le package $scikit-learn$ (nom d'import $sklearn$) pour la classification de chiffres.

1. Le jeu de données `digits` contient des images de chiffres numérisés. On va se servir de ces données pour traiter un problème de classification supervisée.

   ```
   # Chargement des données disponible dans le package sklearn
   from sklearn import datasets
   digits = datasets.load_digits()
   #(vous pouvez aussi écrire : from sklearn.datasets import load_digits)
   # Chargement dans les vecteurs X et y

   X, y = digits.data, digits.target
   ```

2. Affichez une ligne de la matrice X (correspond à un chiffre, i.e. une image de taille 8x8)
3. En changeant l'indice de X, affichez une ligne quelconque de la matrice
4. Visualisez ce chiffre. Vous pourrez utiliser la commande suivante :

   ```
   plt.imshow(np.reshape(X[idx_to_test, :], (8, 8)));
   # vérifiez ce que fait imshow et reshape
   ```

5. Affichage en niveaux de gris. Vous pouvez améliorer la visualisation en tapant les lignes de code suivantes :

```
plt.imshow(np.reshape(X[idx, :], (8, 8)),
           cmap='gray', aspect='equal', interpolation='nearest')

plt.title('Le chiffre numéro %s est un %s' % (idx, y[idx]));
# avec idx l'indice de l'exemple choisi dans X
```

6. Apprentissage par régression logistique.
   On procède de manière classique en réservant 80% des données pour la partie apprentissage, et 20% pour l'évaluation des classifieurs que l'on a construit sur la première partie. En effet il n'est pas raisonnable de tester la performance sur 100% des données. Cela donnerait lieu à du sur-apprentissage (en anglais : *overfitting*). La généralisation des méthodes apprises serait alors très mauvaise.

   ```
   from sklearn.model_selection import train_test_split
   # Découper (split) les données en training) / test sets
   X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
                                                       random_state=42)
   print("Nb d'échantillons d'apprentissage :  {}".format(X_train.shape[0]))
   print("Nb d'échantillons de validation :    {}".format(X_test.shape[0]))
   ```

7. Afficher l'histogramme des valeurs des chiffres

   ```
   plt.hist(y_train, bins=10)
   plt.hist(y_test, bins=10)
   ```

8. Application de la classification supervisée

   ```
   from sklearn.linear_model import LogisticRegression
   # Créer un objet linear regression
   clf_logit = LogisticRegression(solver = 'newton-cg', max_iter=100, multi_class='auto')
   # Entraîner (Train) le modèle à  partir des ensembles d'apprentissage (training sets)
   clf_logit.fit(X_train, y_train)

   # Prédire à  partir des ensembles de test (test sets)
   y_pred_logit = clf_logit.predict(X_test)
   ```

   Vérifiez la signification des différents paramètres (solvers: 'bfgs', 'newton-cg', ... ; nombre d'itérations,...). Par défaut, le solver 'bfgs' est sélectionné.

9. Mesures de performance - Accuracy

   ```
   # Chargement d'une mesure standard de performance
   from sklearn.metrics import accuracy_score
   # Pourcentages des bonnes prédictions
   print("Accuracy logit       : ", accuracy_score(y_test, y_pred_logit))
   print("Accuracy bis logit:  : ", np.mean(y_test == y_pred_logit)) # mesure d'erreur 0/1
   print("Le classifieur logit propose une bonne prédiction dans %s %% des cas."
         % (100 * accuracy_score(y_test, y_pred_logit)))
   ```

10. Mesures de performance pour la classification - Precision, recall and F-measures
    La fonction `classification_report` permet de fournir certaines de ces mesures. Voir la documentation des fonctions `recall_score` et `precision_score` sur le site de `scikit-learn`.

    ```
    from sklearn.metrics import classification_report, f1_score
    print classification_report(y_test, y_pred_logit)
    ```

11. Mesure de performance - Matrice de confusion
    Voir la documentation des fonctions `confusion_matrix` sur le site de `scikit-learn`.

    ```
    from sklearn.metrics import confusion_matrix
    conf_mat_logit = confusion_matrix(y_test, y_pred_logit)
    print("Matrice de confusion pour le classifieur logit : ")
    print(y_pred_logit)
    ```

12. Vous suivrez la même démarche pour le classifieur k-NN (classification par plus proches voisins). L'idée est très simple : pour un nouveau chiffre, on prédit la classe dont le chiffre moyen est le plus proche.

    ```
    from sklearn.neighbors import KNeighborsClassifier
    clf_knn = KNeighborsClassifier(n_neighbors=1)
    ```

13. Comparez les résultats de classification obtenus.