

UNIVERSITÉ DE STRASBOURG
INTELLIGENCE ARTIFICIELLE
Licence 3 - UFR Mathématique - Informatique / Printemps 2020
Projet – Apprentissage bout-en-bout sur des données
Analyse de données synthétiques par Réseau de Neurones Artificiel

Instructions :

- Vous pouvez travailler en binôme (pas de trinôme et plus tolérés)
- Vous devrez rendre votre code et un compte rendu de 2 pages maximum au format **pdf** avant le 15 mai 2020 à 20h CET, dans l'espace Moodle prévu à cet effet (1 rendu par groupe)
- La construction du modèle et son analyse sont des parties indépendantes : je vous fourni des prédictions produites par mon modèle et les données de test correspondantes (*i.e.* si vous ne parvenez pas à obtenir des résultats avec votre modèle, vous pouvez donc quand même faire toute la partie analyse)

Barème (donné à titre indicatif)

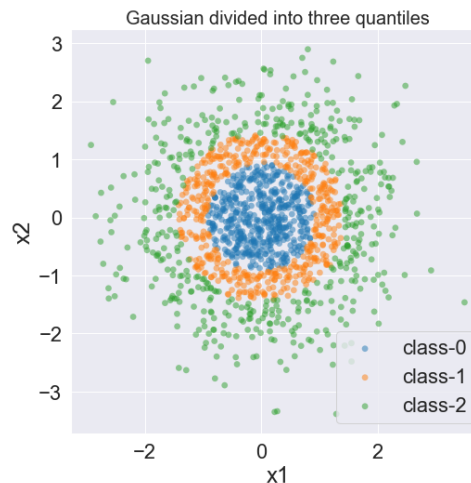
- Préparation des données : 4 points
- Construction et entraînement du modèle : 8 points
- Analyse des résultats : 8 points

1 Préparation des données

Les données à utiliser dans ce travail sont des données synthétiques générées avec Sklearn¹. Comme, lors des TP, vous avez déjà appris à charger des données d'un fichier `csv`, c'est sous ce format que je vous propose les données (le fichier est joint au sujet sur Moodle `gaussian_data.csv`).

x1	x2	class
0.4288123146206977	0.197300419992852	class-0
1.0119033118267453	0.3210523333216393	class-1
-1.2915111010958051	0.42709731679166757	class-1
0.7957031416997422	0.6186639374806492	class-1
0.5032846688166488	0.8130736316294986	class-1
0.6771468663576308	-0.24530054133253051	class-0
0.26429782197543344	0.7251994726731844	class-0
0.7454041973454018	-0.8859159959592792	class-1
1.2736403321516463	-0.4982678831003497	class-1
-0.7055540917782499	1.008001947245045	class-1
0.8735164244697508	-1.4922687866598638	class-2

(a) Aperçu du fichier csv



(b) Visualisation des données

FIGURE 1 – Aperçu des données

🔗 Questions

1. Combien d'attributs ces données comportent-elles ?
2. En combien de classes ces données sont-elles séparées ?
3. Ces données sont-elles linéairement séparable ?

💻 Chargement et préparation des données

1. Chargez les données dans un dataframe Pandas : `gaussian_df = pandas.read_csv(...)`
2. Avant d'aller plus loin, nous allons tout de suite mettre de côté une portion des données pour le test final de notre modèle (qui nous permettra de réaliser l'analyse du modèle en partie 3) :

```
test_final_df = gaussian_df.sample(frac=0.2, random_state=42)
gaussian_df = gaussian_df.drop(test_final_df.index)
```
- (a) Utiliser ce code dans votre programme.
- (b) Expliquer ce que les variables `test_final_df` et `gaussian_df` contiendront suite à l'exécution de ces instructions.
3. Séparez les attributs des classes du dataframe `gaussian_df` : mettez les attributs dans une variable `X` (resp. `X_test_final`) et les classes dans une variable `y` (resp. `y_test_final`).
4. Encodage en one-hot les classes de `y` (resp. `y_test_final`)
5. Enfin, utilisez la méthode de Sklearn `train_test_split()` sur `X` et `y` afin de créer un jeu d'entraînement et un jeu de test.

1. Les curieux peuvent consulter la page suivante pour en savoir plus : https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_gaussian_quantiles.html

2 Construction du modèle

2.1 Adaptation de la méthode `__feed_forward(self, X, y)`

Pour l'analyse du modèle dans la partie suivante, nous aurons besoin de récupérer les sorties calculées par notre modèle (en plus de l'erreur, qui est normalement déjà renvoyée par votre fonction) : modifiez votre fonction de passe avant pour que celle-ci renvoie un couple `erreur, self.A[L]` où `self.A[L]` est l'activation sur la couche de sortie de votre modèle. Pensez, dans votre méthode `fit()` à adapter les instructions où la fonction de passe avant est appelée ! (pour récupérer l'erreur, il faut accéder au premier élément du couple)

2.2 Une méthode pour faire des prédictions

Mettez en œuvre la méthode `def predict(self, x)` permettant de renvoyer la sortie du modèle pour l'instance `x` (il faut utiliser la méthode de passe avant et accéder au second élément du couple).

2.3 Le modèle à entraîner

Vous allez définir, en utilisant le modèle de réseau de neurones conçu à l'occasion des TP, un modèle à 3 couches cachées de respectivement 4, 3 et 2 unités. Les unités seront activées avec une tangente hyperbolique et le réseau sera entraîné pendant 200 époques avec un pas d'apprentissage $\eta = 0.01$ et des *batch* de taille 1 (comme lors des TP).

(Notez que l'entraînement peut prendre un peu de temps...).

🔧 Combien de paramètres (poids et biais) comporte ce modèle ?

🔧 Que pourrait-il se passer si on augmentait le pas d'apprentissage, par exemple à la valeur $\eta = 0.1$?

🖥️ Entraînez votre modèle selon les instructions ci-dessus et afficher avec Matplotlib l'évolution de l'erreur sur le jeu d'entraînement et sur le jeu de test (exemple illustré sur la figure 2)



FIGURE 2 – Exemple d'évolution des erreurs sur les données d'entraînement et de test

3 Analyse du modèle

Dans cette partie, vous allez utiliser les prédictions faites par votre modèle (dénnotées par la suite `y_pred`) sur les données `X_test_final` afin d'étudier le comportement du modèle en comparant ces prédictions avec les résultats effectivement attendus (`y_test_final`).

Pour obtenir des prédictions, il faut appeler la méthode `predict()` sur chaque instance de `X_test_final` :

```
y_pred = []
for i in range(0, len(X_test_final)):
    y_pred.append(model.predict(X_test_final[i])[1].transpose())
```

Les predictions seront stockées dans une matrice de la forme :

```
# Affichage avec l'instruction: print(y_pred[:5])
[[0.00 0.17 0.83] # label prédit de l'instance X_test_final[0]
 [0.00 0.19 0.81] # label prédit de l'instance X_test_final[1]
 [0.00 0.12 0.88] # label prédit de l'instance X_test_final[2]
 [0.97 0.03 0.00] # label prédit de l'instance X_test_final[3]
 [0.98 0.02 0.00]] # label prédit de l'instance X_test_final[4]
```


Et on va chercher à les comparer aux vraies étiquettes :


```
# Note: pour faciliter la manipulation, les étiquettes doivent être converties en tableau Numpy:
# y_actual = y_test_final.to_numpy()
[[0.00 0.00 1.00] # vrai label de l'instance X_test_final[0]
 [0.00 0.00 1.00] # vrai label de l'instance X_test_final[1]
 [0.00 0.00 1.00] # vrai label de l'instance X_test_final[2]
 [1.00 0.00 0.00] # vrai label de l'instance X_test_final[3]
 [1.00 0.00 0.00]] # vrai label de l'instance X_test_final[4]
```

3.1 Ratio *prédictions correctes* / *total prédictions*

Examinons la prédiction pour `X_test_final[0]` :

`y_pred[0] = [0.00 0.17 0.83]` et `y_actual[0] = [0.00 0.00 1.00]`.

 Quelle classe est prédite pour l'instance $i = 0$? Cette prédiction est-elle correcte?

 En utilisant la fonction `max_row(row)` définie ci-dessous, calculer, pour l'ensemble des données de test final, le pourcentage de prédictions correctes réalisées par votre modèle. Quel pourcentage avez-vous obtenu?

```
def max_row(row):
    """
    row: 1D array
    Returns: index of the maximum value in row
    """
    return np.where(row == np.amax(row))[0][0]
```


3.2 Matrice de confusion

En cours, nous avons vu comment décompter les vrais/faux positifs/négatifs pour construire une matrice de confusion. On propose ici de généraliser ce principe pour un nombre de classes > 2 .

Initialement, vous aurez une matrice remplie de zéros, de dimensions $c \times c$ (ou c est le nombre de classes à examiner)². Ensuite, pour l'ensemble des n instances de `X_test_final`, il faudra décompter combien de :

- `class-0` ont été prédits comme `class-0`;
- `class-0` ont été prédits comme `class-1`;
- `class-0` ont été prédits comme `class-2`;
- `class-1` ont été prédits comme `class-0`;
- `class-1` ont été prédits comme `class-1`;
- *etc.*


et écrire ces résultats dans la matrice de confusion.

 Mettez en œuvre le code nécessaire à la création de la matrice de confusion pour les prédictions de votre modèle.

Note : vous pouvez afficher votre matrice avec le code suivant :

```
import matplotlib.pyplot as plt
import seaborn as sns
class_names = ['class-0', 'class-1', 'class-2']
plt.figure(figsize = (8,8))
sns.set(font_scale=2) # label size
ax = sns.heatmap(confusion_mtx, annot=True, annot_kws={"size": 30}, # font size
                  cbar=False, cmap='Blues', fmt='d', # format (int)
                  xticklabels=class_names, yticklabels=class_names)
ax.set(title='', xlabel='Actual', ylabel='Predicted')
plt.show()
```

 Examinez la matrice obtenue et expliquer l'erreur la plus fréquente de votre modèle.

 Expliquer pourquoi cette analyse complète bien la métrique “pourcentage de prédictions correctes” calculée ci-dessus.

Ocazou : des données pour l'analyse

Si des fois vous n'avez pas pu obtenir de données par votre modèle, vous pouvez quand même faire toute la partie 3 en utilisant les deux fichiers `432-e200-b1-eta0.01-tanh_y_actual.csv` et `432-e200-b1-eta0.01-tanh_y_pred.csv` à disposition sur Moodle. Pour les lire et écrire dans des tableaux Numpy les données contenus dans ces fichiers, utilisez les instructions :

```
y_actual = np.genfromtxt('432-e200-b1-eta0.01-tanh_y_actual.csv', delimiter=',')
y_pred = np.genfromtxt('432-e200-b1-eta0.01-tanh_y_pred.csv', delimiter=',')
```

2. Pour initialiser une matrice numpy avec des zéros : <https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html>