



UNIVERSITÉ DE SHERBROOKE

PROJET IFT 712
INFORMATIQUE - DEUXIÈME CYCLE

MÉTHODES DE CLASSIFICATION PAR SKLEARN

Réalisé par :
THOUIN Kevin
BENNANI Kaoutar

Supervisé par :
Pr. Pierre-Marc JODOIN

Année universitaire 2019 - 2020

Table des matières

Table des figures	2
1 Présentation du projet	3
1 Présentation	3
2 Choix de la base de données	3
3 Choix du design	3
4 Gestion du projet	4
2 Les algorithmes et démarche utilisés	5
1 Algorithmes	5
2 Démarche scientifique	5
2.1 Organisation des données	5
2.2 Cross-validation	6
2.3 Recherche des hyper-paramètres	6
3 Analyse des résultats	7
1 Résultats	7
1.1 Régression logistique	7
1.2 SVM	8
1.3 Réseaux de neurones	8
1.4 Bagging	9
1.5 AdaBoost	10

Table des figures

3.1	Figure diagrames "accuracy" et "loss" pour la régression logistique	7
3.2	Figure parametres de régression logistique	8
3.3	Figure SVM-error	8
3.4	Figure diagrames "accuracy" et "loss" pour réseau de neuronne	8
3.5	Figure paramètres pour une fonction d'activation logistique	9
3.6	Figure paramètres pour une fonction d'activation RELU	9
3.7	Figure bagging-param	9
3.8	Figure bagging-error	9
3.9	Figure adaboost-param	10
3.10	Figure adaboost-error	10

Chapitre 1

Présentation du projet

1 Présentation

Ce projet de session fait partie des travaux du cours IFT712, il a pour objectif de tester quelques méthodes de classification sur une base de données Kaggle (www.kaggle.com) avec la bibliothèque Sklearn .

2 Choix de la base de données

Nous avons choisi comme base de données : "Wine Dataset".

A propos de la base de données, les données sont le résultat d'une analyse chimique de vins cultivés dans une des régions d'Italie mais issus de trois cultivars différents.

Parmi ses caracteristiques :

- 178 instances
- 13 variables
- les valeurs des attributs sont des Integer et des Float
- Pas de valeurs manquantes
- Généralement utilisé pour les tâches de classification

3 Choix du design

Le projet contient 8 classes en total.

- La classe 'Classification_main.py'
- La classe 'Classification_neural_net.py' : Décrit l'algorithme de classification par réseau de neurones.
- La classe 'Classification_logistique.py' : S'agit de la classification par régression logistique.
- La classe 'Classification_bagging.py' : Classification par l'algorithme Bagging.
- La classe 'Classification_adaboost.py' : Classification par l'algorithme AdaBoost.
- La classe 'Classification_svm.py' : Il s'agit du code correspondant au SVM.

- La classe 'Classification_hyperparameter.py' : Il s'agit des techniques de la recherche des hyperparamètres pour chacun des algorithmes de classification.
- La classe 'Classification_io.py' : Cette classe crée des données d'entraînement et de test à partir du "Wine Dataset" et a une fonction permettant l'affichage graphique.

4 Gestion du projet

Pour la réalisation du projet, nous avons utilisé le gestionnaire de version de code "git" via la plateforme "gitHub"

Chapitre 2

Les algorithmes et démarche utilisés

Cette partie consiste à présenter les algorithmes utilisés, ainsi que la démarche scientifique suivie.

1 Algorithmes

Les algorithmes utilisés sont :

- Régression logistique : les hyperparamètres sont le terme de régularisation et le taux d'apprentissage. Cette algorithme devrait bien performé si les données sont linéairement séparable, ce qui est une hypothèse peut-être audacieuse.
- SVM : Nous utilisons le noyau rbf. Nous n'avons pas d'hyperparamètres pour cette algorithme de classification.
- Réseaux de neurones : Nous testons avec une ou deux couches de six neurones. Les hyperparamètres sont la fonction d'activation (relu ou logistique), le terme de régularisation, le taux d'apprentissage et le momentum pour la descente de gradient.
- Bagging : Nous avons deux hyperparamètres. Le premier est l'estimateur (adaboost ou un réseau de neurones avec de couches de six neurones), le second est le nombres d'estimateurs
- AdaBoost : Nous utilisons un arbre de décision de profondeur un comme estimateur de base. Nous avons deux hyperparamètres. Le premier est le nombre d'estimateurs et le second est le taux d'apprentissage

2 Démarche scientifique

2.1 Organisation des données

Après la récupération des données du "Wine Dataset", on a divisé ces données en des données d'entraînement et des données de test. Cela se voit dans la classe : "Classification_io.py"

Donc, l'entraînement et le test ont été fait sur deux données différentes : `x_train` et `x_test`.

2.2 Cross-validation

2.3 Recherche des hyper-paramètres

La classe "Classification_hyperparameter.py" est notre implémentation de la méthode de la recherche des hyper-paramètres. Ceci a été fait à l'aide de "GridSearchCV" de la bibliothèque "sk-learn.model_selection"

Chapitre 3

Analyse des résultats

Dans cette partie nous allons analyser les résultats de chacun des algorithmes.

1 Résultats

1.1 Régression logistique

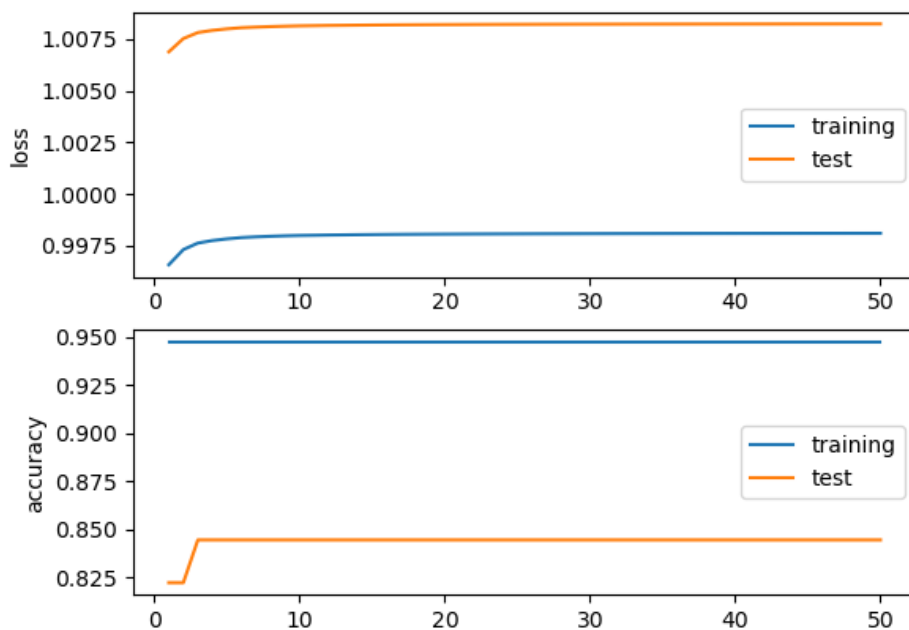


FIGURE 3.1 – Figure diagrammes "accuracy" et "loss" pour la régression logistique

Pour " lr " entre 0.0001 et 0.001, et pour " l2reg " entre 0.1 et 10, les meilleurs valeurs pour les hyper-paramètres " lr " et " l2reg " sont ceux affichés dans la figure ci-dessous


```

logistique_param - Bloc-notes
Fichier Edition Format Affichage Aide
12reg: 7.800000000000001
1r: 0.00091818181818182

```

FIGURE 3.2 – Figure parametres de régression logistique

1.2 SVM

Les erreurs et les précisions d'entraînement et de test :

```

svm_error - Bloc-notes
Fichier Edition Format Affichage Aide
Erreur d'entraînement : 0.7082
Erreur de test : 0.3997
Précision d'entraînement : 0.6992
Précision de test : 0.8222

```

FIGURE 3.3 – Figure SVM-error

1.3 Réseaux de neurones

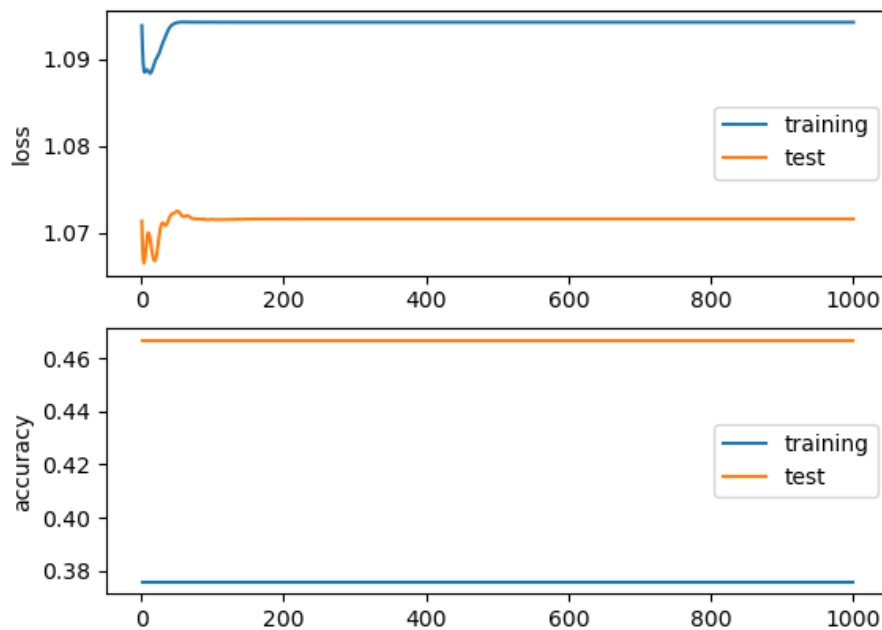
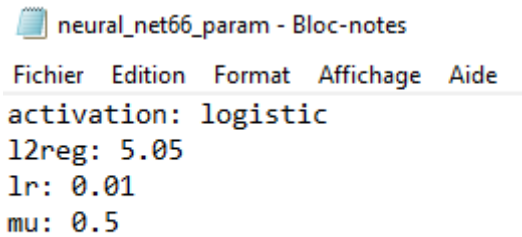


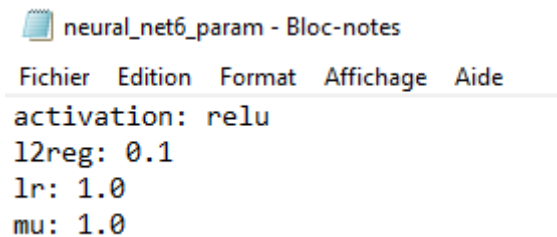
FIGURE 3.4 – Figure diagrames "accuracy" et "loss" pour réseau de neuronne

Pour "lr" entre 0.01 et 1, et pour "l2reg" entre 0.1 et 10, et pour "mu" entre 0 et 1, les meilleurs valeurs pour les hyper-paramètres "lr", "l2reg" et "mu" sont ceux affichés dans les deux figures ci-dessous



```
neural_net66_param - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
activation: logistic
l2reg: 5.05
lr: 0.01
mu: 0.5
```

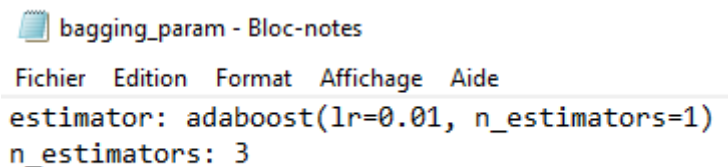
FIGURE 3.5 – Figure paramètres pour une fonction d'activation logistique



```
neural_net6_param - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
activation: relu
l2reg: 0.1
lr: 1.0
mu: 1.0
```

FIGURE 3.6 – Figure paramètres pour une fonction d'activation RELU

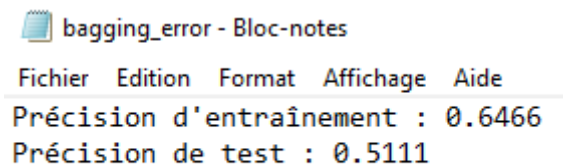
1.4 Bagging



```
bagging_param - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
estimator: adaboost(lr=0.01, n_estimators=1)
n_estimators: 3
```

FIGURE 3.7 – Figure bagging-param

Les précisions d'entraînement et de test :

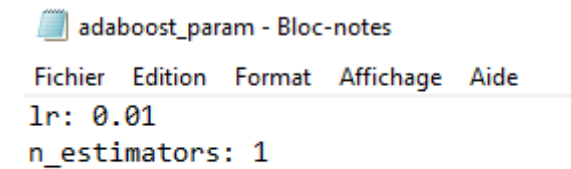


```
bagging_error - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
Précision d'entraînement : 0.6466
Précision de test : 0.5111
```

FIGURE 3.8 – Figure bagging-error

1.5 AdaBoost

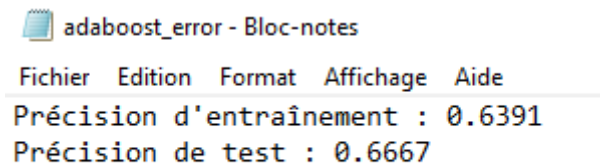
Pour "lr" entre 0.01 et 1, et pour "n-estimators" entre 1 et 101, les meilleures valeurs pour les hyper-paramètres "lr" et "n-estimator" sont ceux affichés dans la figure ci-dessous



```
adaboost_param - Bloc-notes
Fichier Edition Format Affichage Aide
lr: 0.01
n_estimators: 1
```

FIGURE 3.9 – Figure adaboost-param

Les précisions d'entraînement et de test :



```
adaboost_error - Bloc-notes
Fichier Edition Format Affichage Aide
Précision d'entraînement : 0.6391
Précision de test : 0.6667
```

FIGURE 3.10 – Figure adaboost-error