CY Cergy Paris Université
Composante Sciences et Techniques
Master d'Informatique - 1ère année

Systèmes Intelligents et Communiquants

Mention Informatique et Ingénierie des Systèmes Complexes

Intelligence Artificielle

TP1 IA+

Rapport

Sterley Gilbert Labady

16 février 2023

1 Introduction

Le but de ce TP est de mettre en oeuvre une classification naïve bayésienne sur deux applications différentes : la reconnaissance visuelle de chiffres manuscrits.

2 Application à la reconnaissance de chiffres

Dans ce premier exercice, nous appliquons l'algorithme sur de la classification naïve bayésienne sur la base de données MNIST.

Comment estimer les paramètres de ce modèle à partir des données d'entraînement ? Tout d'abord, nous avons chargé les données d'entraînement MNIST en utilisant la fonction *mnist.load_ data()*, et nous avons remodelé les images en vecteurs de pixels pour faciliter la manipulation des données. Nous avons ensuite créé une classe BayesianNumberClassifier pour implémenter la classification naïve bayésienne. Les paramètres du modèle de classification naïve bayésienne sont les tableaux de probabilités conditionnelles *histogram*, *prob_ class_ tab*, et *prob_ pixel_ i_ sg_ tab*, qui sont estimés à partir des données d'entraînement.

- *histogram* : La méthode *train* de la classe *BayesianNumberClassifier* est utilisée pour entraîner le modèle en estimant les probabilités des niveux de gris pour chaque pixels pour chaque classe. Ces probabilitées sont ensuite stockées dans un tableau de probabilités conditionnelles appelé *histogram*.
- *prob_ class_ tab* : Les probabilités a priori de chaque classe sont également estimées en utilisant les données d'entraînement. Cela est réalisé en comptant le nombre d'images de chaque classe dans l'ensemble d'entraînement, et en divisant ce nombre par le nombre total d'images dans l'ensemble d'entraînement pour obtenir la probabilité a priori de chaque classe. Ces probabilités sont stockées dans un tableau appelé *prob_ class_ tab*.
- *prob_ pixel_ i_ sg_ tab* : La méthode *get_ prob_ pixel_ i_ sg_ tab* de la classe *BayesianNumberClassifier* est utilisée pour estimer les probabilités conditionnelles des niveaux de gris des pixels pour chaque classe. Pour chaque pixel i de chaque classe c, la méthode utilise la formule suivante : la probabilité que le pixel i ait une intensité sg étant donné que l'image appartient à la classe c est égale à la fréquence du nombre d'images de la classe c où le pixel i a une intensité sg, divisé par le nombre total d'images dans la classe c. Ces probabilités conditionnelles sont stockées dans un tableau 2D appelé *prob_ pixel_ i_ sg_ tab*.

Implémenter l'estimation des paramètres du modèle Pour estimer ces paramètres, la classe utilise les méthodes suivantes :

- *generate_ histo* pour générer des histogrammes de chaque pixel pour chaque classe.
- *get_ prob_ class_ c* pour estimer les probabilités a priori de chaque classe.
- *get_ prob_ pixel_ i_ sg* pour estimer les probabilités conditionnelles de niveau de gris de chaque pixel.
- *get_ prob_ class_ tab* pour stocker les probabilités a priori de chaque classe dans un tableau.
- *get_ prob_ pixel_ i_ sg_ tab* pour stocker les probabilités conditionnelles de niveau de gris de chaque pixel pour chaque classe dans un tableau.

Une fois que ces paramètres ont été estimés à partir des données d'entraînement, le modèle peut être utilisé pour prédire la classe d'une nouvelle image en calculant la probabilité logarithmique pour chaque classe donnée, en utilisant les paramètres estimés. La classe de la probabilité la plus élevée est choisie comme la prédiction du modèle.

Implémenter une fonction classifiant une entrée x en fonction des paramètres du modèle La fonction *predict* est la fonction clé de la classe *BayesianNumberClassifier*. Elle prend en entrée un tableau d'intensités de pixels pour une image, calcule la probabilité que l'image appartienne à chaque classe, et retourne la classe avec la plus haute probabilité.

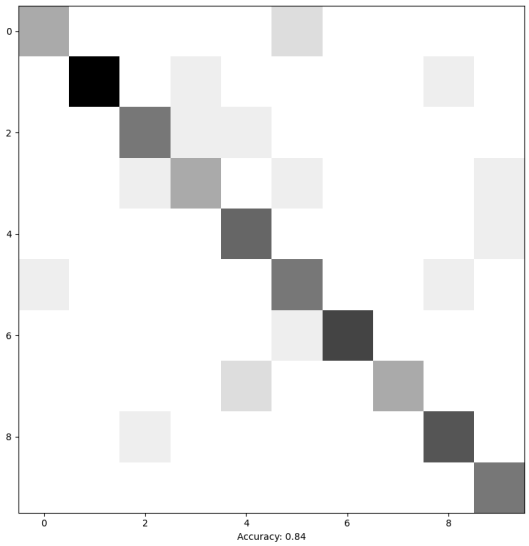

FIGURE 1 – Accuracy : 0.84

Affichage de la matrice de confusion obtenue après apprentissage

Quelle est la précision de l'algorithme ? L'accuracy obtenue sur le jeu de test de 100 exemples tirés aléatoirement pour la matrice de confusion est de 0.84, et pour tous les 10000 exemples du jeu de test mnist il est de 0.825, ce qui indique que le modèle est capable de prédire correctement la classe d'environ 83% des images du jeu de test.

Afficher la matrice de paramètres *p(x/yk)* correspondant à chaque classe *yk* et commenter

```
# Accédez à la matrice d'histogrammes
histogram = bayes.histogram
# Normalisez chaque histogramme pour chaque classe
total_images = len(train_X)
normalized_histogram = histogram / total_images
# Empilez les histogrammes dans une matrice 2D
p_x_yk = np.vstack(normalized_histogram)
# Affichez la matrice de paramètres pour chaque classe
print("Matrice de paramètres p(x|yk) pour chaque classe : \n", p_x_yk)
```

3 Application à la détection de spam

Dans ce deuxième exercice, on applique l'algorithme sur de la classification binaire sur une base de données de SMS, téléchargeable sur `https://archive.ics.uci.edu/ml/datasets/sms+spam+collection`

Comment estimer les paramètres de ce modèle à partir des données d'entraînement ? Les paramètres du modèle sont estimés à partir des données d'entraînement de la manière suivante :

— Tout d'abord, trois dictionnaires vides sont initialisés pour stocker les probabilités estimées : *self.class_ prob_ dico* pour stocker la probabilité de chaque classe, *self.words_ occ_ dico* pour stocker le nombre de fois où chaque mot apparaît dans le corpus d'entraînement, et *self.words_ occ_ dico_ byClass* pour stocker le nombre de fois où chaque mot apparaît dans chaque classe.

— Ensuite, pour chaque classe dans le corpus d'entraînement, la probabilité de cette classe est estimée en comptant le nombre de messages dans cette classe et en le divisant par le nombre

total de messages dans l'ensemble d'entraînement. La probabilité de la classe est stockée dans le dictionnaire *self.class_prob_dico*.
— Pour chaque message dans l'ensemble d'entraînement, chaque mot dans le message est ajouté au dictionnaire *self.words_occ_dico*, en comptant le nombre de fois où chaque mot apparaît dans l'ensemble d'entraînement.
— Pour chaque mot dans chaque message, le nombre de fois où ce mot apparaît dans la classe correspondante est ajouté au dictionnaire *self.words_occ_dico_byClass*. Si le mot n'a pas encore été ajouté au dictionnaire, il est ajouté avec une valeur de 1. Sinon, la valeur correspondante est incrémentée de 1.
— Une fois que les comptages de mots pour chaque classe ont été effectués, les probabilités conditionnelles des mots étant dans chaque classe sont calculées à partir des comptages stockés dans le dictionnaire *self.words_occ_dico_byClass*. La probabilité conditionnelle d'un mot étant dans une classe est le nombre de fois que le mot apparaît dans cette classe, divisé par le nombre total de mots dans cette classe.

Implémenter l'estimation des paramètres du modèle Dans le code fourni, les paramètres du modèle sont calculés à partir des données d'entraînement lors de l'appel à la méthode train. Cela implique une analyse en temps réel des données pour déterminer la probabilité de chaque classe et la probabilité conditionnelle de chaque mot étant donné une classe. Cependant, ces calculs peuvent être coûteux en temps de traitement si l'ensemble de données est très grand ou s'il y a des centaines de milliers de mots différents. Pour éviter de recalculer ces paramètres à chaque fois que le modèle est exécuté, une alternative est de les enregistrer dans un fichier après le premier entraînement, et de les charger dans le modèle lors des entraînements ultérieurs. Cela peut être fait en utilisant le module *pickle* de Python pour enregistrer les dictionnaires *class_prob_dico*, *words_occ_dico* et *words_occ_dico_byClass* dans un fichier binaire et les charger à partir du même fichier lors de l'exécution ultérieure du modèle. Cette technique permet de gagner du temps de traitement lors des exécutions ultérieures du modèle.

Implémenter une fonction classifiant une entrée x en fonction des paramètres du modèle La fonction *predict* implémentée dans la classe *BayesianSpamClassifier* est utilisée pour prédire la classe (*ham ou spam*) d'un message donné en entrée, en utilisant les probabilités calculées pour chaque classe à partir des données d'entraînement. La fonction prend en entrée un message représenté sous forme d'une liste de mots, et retourne la classe prédite pour ce message. Pour chaque classe c, la fonction calcule un score de probabilité en utilisant la formule de Bayes, en combinant la probabilité de la classe c avec les probabilités conditionnelles de chaque mot du message étant donné la classe c, et avec les probabilités marginales de chaque mot. La probabilité de la classe c est obtenue à partir du dictionnaire *class_prob_dico*, tandis que les probabilités conditionnelles et marginales de chaque mot sont obtenues à partir des dictionnaires *words_occ_dico_byClass* et *words_occ_dico*, respectivement. La fonction renvoie finalement la classe c avec le score de probabilité le plus élevé.

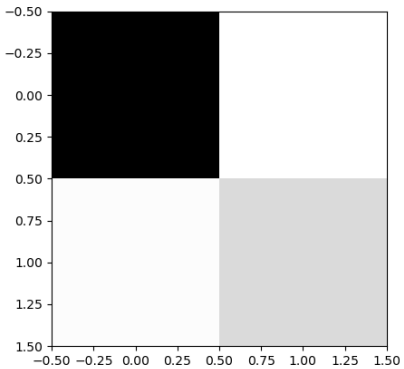

FIGURE 2 – Affichage de la matrice de confusion obtenue après apprentissage | Accuracy : 0.98

Quelle est la précision de l'algorithme ? L'accuracy obtenue sur le jeu de test de 1115 exemples est de 0.98, ce qui indique que le modèle est capable de prédire correctement la classe d'environ 98% des messages du jeu de test.

Afficher les mots ayant la plus forte probabilité d'apparaître dans du spam Les cinq mots les plus probables d'apparaître dans les messages de spam sont "to", "a", "your", "call" et "or".