M1 Informatique - Intelligence Artificielle

Travaux Pratiques

HUBERT Capucine
SUCHAUD Florian

Janvier 2022

1 TP1 : Implémentation des méthodes des K plus proches Voisins et du Perceptron en Python

1.1 L'algorithme des k plus proches voisins

1.1.1 Principe général

La méthode des k plus proches voisins est une méthode d'apprentissage supervisée. Le succès de cette méthode dépend en partie de la qualité des données d'entraînement ainsi que de la méthode de calcul de la mesure de distance. La plupart du temps cette distance va correspondre à la distance Euclidienne calculée comme suit avec X un vecteur et x un scalaire.

$$d(X_1, X_2) = \sqrt{\sum_k (x_{1,k} - x_{2,k})^2}$$

Cette méthode des k plus proches voisins consiste à regarder, parmi les exemples d'apprentissage, les classes associées aux k entrées X_t qui sont les plus "proches" (au sens de la distance Euclidienne) de l'entrée donnée X. Et ainsi d'associer à celle-ci la classe majoritaire retourner par ses k plus proches voisins.

1.1.2 Algorithme en Python

Dans notre algorithme nous récupérons tout d'abord l'ensemble des données d'apprentissage ainsi que celles à classées. Nous calculons ensuite la distance séparants les points entre eux. Seules les classes correspondant aux k plus proches voisins sont récupérées. Et enfin, nous associons à notre donnée X la classe obtenue majoritairement.

Afin d'améliorer notre algorithme, nous rajoutons à la liste d'apprentissage les données et leur classe associée au fur et à mesure de la classification. Nous affichons ensuite le résultat graphiquement faisant apparaître les points de couleurs différentes selon leur classe. Et afin de nous assurer de la fiabilité de notre algorithme nous calculons le taux d'erreur de l'algorithme en comparant ce que nous obtenons aux résultats attendus.

1.2 L'algorithme de mise en place d'un perceptron simple

1.2.1 Principe général

La méthode du perceptron est, tout comme la méthode précédente, une méthode d'apprentissage supervisée. Cette méthode peut modéliser la décision de la classe d'un vecteur d'entrée à l'aide de différentes fonctions. Il peut s'agir d'une

fonction linéaire suivi d'un seuil qui détermine la classe du vecteur d'entrée ou encore d'une régression logistique qui prédit la probabilité de ce vecteur d'appartenir à la classe 0 ou 1.

$H_w(X) = Threshold(W.X)$

où : $Threshold(z)=1$ si $z>=0$ et 0 sinon

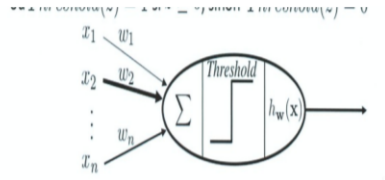

FIGURE 1 – Fonction linéaire suivi d'un seuil

$$p(y = 1|\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

FIGURE 2 – Régression logistique

Le principe d'apprentissage de ce type de perceptron est d'adapter les valeurs des paramètres du modèle, qui correspondent aux vecteurs de poids W. En effet, les poids et le biais sont mis à jour au cours des itérations, à l'aide notamment du taux d'apprentissage, lorsque la classe retournée ne correspond pas à la véritable classe associée aux valeurs d'apprentissage. Cette mise à jour des poids est appelée la règle d'apprentissage du perceptron. Celle-ci est arrêtée lorsque le nombre d'erreur est égale à 0 ou lorsque l'apprentissage a atteint un nombre maximal d'itérations. Le perceptron cherche ainsi un séparateur linéaire entre les deux classes qu'il est possible de représenter graphiquement.

1.2.2 Algorithme en Python

Il faut tous d'abord savoir que dans l'algorithme que nous allons implémenter nous allons utiliser la régression logistique plutôt que la fonction linéaire. En effet, cette dernière n'utilisant pas les probabilités d'appartenir à une classe mais la déterminant directement est moins précise que la fonction sigmoïde que nous allons alors utiliser.

Dans notre algorithme nous implémentons tout d'abord le perceptron simple à un seul neurone. Nous vérifions ainsi la fonction d'activation à utiliser. Ensuite nous passons à l'implémentation de la fonction d'apprentissage. Pour cela nous prenons des poids synaptiques ainsi qu'un biais qui pourrons ensuite être modifiés après comparaison du résultat entre la classe calculée et la classe attendue.

Afin d'améliorer notre algorithme, nous décidons d'utiliser 100 itérations de l'apprentissage. Nous affichons ensuite le résultat graphiquement faisant apparaître les points de couleurs différentes selon leur classe ainsi que le séparateur linéaire associé afin de nous assurer de la fiabilité de notre algorithme. Nous calculons également le nombre erreurs de l'algorithme en comparant ce que nous obtenons

par rapport aux résultats attendus. Nous affichons ce nombre dans un graphique en fonction du nombre d'itérations. Ainsi, nous pouvons observer la progression de ce nombre au fur et à mesure de l'apprentissage de notre perceptron. Nous observons d'ailleurs qu'il est toujours très vite décroissant durant les premières itérations puis que celui-ci reste constant ensuite.

1.3 Comparaison des résultats

Après exécution de notre premier programme qui utilise les k plus proches voisins, nous observons un taux d'erreur en général égale à 0%. Ainsi avec un k égale à 3, l'algorithme semble toujours obtenir le résultat attendu. Cependant lorsque nous exécutons notre second programme mettant en place un perceptron simple, nous observons que le taux d'erreur, même après les 100 itérations n'est pas toujours égale à 0%, il est régulièrement égale à 4% (observations effectuées sur une trentaine de répétitions). Nous observons en effet que lors de certains de nos essais, un point appartenant à une certaine classe se retrouvait du mauvais côté du séparateur linéaire.

Ainsi nous préconisons ici plutôt l'utilisation du modèle des k plus proches voisins dont le résultat est plus fiable et dont le code est plus simple à implémenter.

2 TP2 : Construction d'un Anti-Spam

2.1 Réseau de neurones

2.1.1 Principe général

Le classifieur basé sur une architecture à base de réseau de neurones avec ou sans couches cachées est une méthode d'apprentissage supervisée. Il se base sur le calcul de l'activité des neurones en fonction des activités des neurones précédents et des poids de chacun d'entre eux selon la courbe de régression logistique vu précédemment dans le perceptron. Ainsi l'activité du j^e neurone noté a_j donne a_j=Logistic($\sum_i w_{i,j} a_i$). Le calcul des poids se fait avec les gradients et le calcul des dérivées partielles. Ce calcul peut notamment être grandement facilité à l'aide de la règle de dérivation en chaînes.

Ainsi, le premier modèle de classifieur que nous verrons en suivant, sans couche cachée, sera constitué de seulement deux neurones, celui d'entrée et celui de sorti. Alors que le deuxième en possédera un de plus qui correspondra au neurone de la couche dite cachée.

2.1.2 Algorithmes en Python

Dans nos algorithmes de réseau de neurones nous récupérons tout d'abord l'ensemble des données d'apprentissage ainsi que celles à classées que nous "apla-

tissons" en vecteurs de 57*57 correspondant au nombre de caractéristiques de chaque données. Ensuite nous créons l'architecture du réseau. Le premier modèle possède deux couches car seulement deux neurones comme expliqué précédemment alors que le second en possède trois. Ensuite il ne faut pas oublier de compiler le modèle de classification multiclasse et d'entraîner les modèles. Pour finir, ceux-ci sont ensuite évalués et le résultat affiché.

2.1.3 Comparaison des résultats des réseaux de neurones

Lorsque nous comparons nos deux modèles de classifieurs basés sur une architecture à base de réseau de neurones nous observons que le réseaux avec une couche cachée obtient de meilleur résultat que le réseau sans couche cachée. En effet, le nombre de calcul effectué, vu précédemment, est plus important et ainsi les résultats plus fiables.

Ainsi, nous préconisons ici plutôt l'utilisation du deuxième modèle qui a obtenu de meilleur résultat et dont le code ne varie que par l'ajout d'une seule ligne de code.

2.2 Réseau de Bayes naïf

2.2.1 Principe général

Le classifieur de type réseau de Bayes naïf est une méthode d'apprentissage supervisée elle aussi. Le modèle probabiliste pour un classifieur Bayésien naïf est un modèle conditionnel qui utilise la règle de Bayes suivante : P(C/X_1,X_2,...X_n)= P(C)*P(X_1,X_2,...X_n/C)/P(X_1,X_2,...X_n). En pratique seul le numérateur est important car le dénominateur ne dépend pas de C et ne va donc pas influencer sur notre résultat qui est une comparaison. De plus, en faisant intervenir l'hypothèse naïve et en supposant chaque X_i indépendants les uns des autres, le classifieur Bayésien naïf nous donne P(X_1,X_2,...X_n/C) = $\prod$P(X_i/C).

Dans un classifieur de type réseau de Bayes naïf cette formule est testée pour l'ensemble des classes. La nouvelle entrée X définie par les différents X_i appartiendra à la classe dont la probabilité P(C/X_1,X_2,...X_n) sera la plus élevée.

Cependant il existe deux cas. Le cas des paramètres discrets et le cas des paramètres continus. Dans le premier cas les différentes probabilités conditionnelles P(X_i/C) sont assez simples à estimer. Cependant, dans le second cas pour calculer ces différentes probabilités on supposera de manière générale que les lois de probabilités correspondantes sont les lois normales dont la formule est la suivante :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Il faudra donc pour cela calculer l'espérance et la variance des X_i comme suit :

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

FIGURE 3 – Calcul de l'espérance

$$\sigma^2 = \frac{1}{(N-1)}\sum_{i=1}^{N} (x_i - \mu)^2$$

FIGURE 4 – Calcul de la variance

2.2.2 Algorithme en Python

Dans notre algorithme de type réseau de Bayes naïf nous récupérons l'ensemble des données comme dans les algorithmes précédents de réseau de neurones. Ensuite, avant de créer l'architecture du réseau, nous stockons les variables que nous allons utiliser pour le calcul des espérances et variances car il s'agit ici de données à paramètres continus et non discrets. Après récupérations de ces valeurs nous les utilisons dans la loi normale afin de pouvoir ensuite récupérer les probabilités conditionnelles qui nous intéressent. Après comparaison entre les deux probabilités associées aux deux classes, nous associons à notre donnée la classe avec la plus forte probabilité. Pour finir nous vérifions les résultats obtenus par rapport aux résultats attendus et nous affichons le taux de similitude.

2.3 Comparaison des résultats

Lorsque nous comparons nos deux modèles précédents de classifieurs basés sur une architecture à base de réseau de neurones et notre classifieur de type réseau de Bayes naïf, nous observons que ce dernier à une efficacité intermédiaire. En effet son taux de réussite se situe entre ceux des deux modèles basés sur une architecture à base de réseau de neurones.

Ainsi, nous préconisons ici plutôt l'utilisation du classifieur basé sur une architecture à base de réseau de neurones à une couche cachée car son résultat est le plus fiable des trois modèles et de plus son code est plus simple à implémenter que celui du modèle Bayésien naïf.