Apprentissage Numérique Automatique

M1 Informatique 2014/15

Miniprojet 1

Holger.Schwenk@lium.univ-lemans.fr

loic.barrault@lium.univ-lemans.fr

L'objectif de ce miniprojet est de programmer un classifieur bayésien avec MATLAB/octave. L'exemple choisi est la classification de chiffres `0` à `9`. Les chiffres sont représentés par des images de taille 16×16 en 256 valeur de gris. Chaque chiffre est donc codé par un vecteur réel de dimension 256.

Le travail demandé consiste en deux phases :
- Apprentissage des paramètres du classifier bayésien avec les données d'apprentissage
- L'utilisation du classifier pour tester ces performances sur les données de test

Il y a 10 000 exemples dans le corpus d'apprentissage (fichier `appr.ascii`), 5 000 dans le corpus de développement (fichier `dev.ascii`) et 10 000 dans le corpus d'évaluation (fichier `eval.ascii`). Les fichiers peuvent être lus par une commande MATLAB du style :

```
A=load("appr.ascii");
```

On obtient une matrice de dimension $10\,000 \times 256$. Chaque ligne contient donc un exemple de dimension 256. La liste des classes de chaque exemple se trouvent dans les fichiers `appr_cl.ascii` et `dev_cl.ascii`. Toutes ces données se trouvent dans le répertoire `/info/projets/m1-ann`.
Évitez de copier ces données, mais utilisez plutôt de liens symboliques.

Pour rappel, voici les étapes à réaliser ;
- Utilisation des données d'apprentissage et leur classe afin de déterminer les paramètres de votre classifieur Bayésien.
- Une fois la première étape terminée, on peut utiliser le système pour classer des données. Pour vérifier le bon fonctionnement, vous utiliserez les données de développement puisque vous en disposez les classes correctes. Ceci permet de calculer le taux d'erreur de votre système.
- Vous pouvez éventuellement répéter les étapes 1 et 2 pour créer différentes variantes de votre système. Vous garderez bien sûr la variante dont le taux d'erreur est minimal.
- Vous appliquez votre meilleur système sur les données d'évaluation. Il faut envoyer un fichier texte à votre professeur avec la classe reconnue pour chaque exemple. C'est lui qui calculera le taux d'erreur.

Ces trois étapes sont détaillées par la suite.

1 Phase d'apprentissage

Il faut estimer deux types de probabilités :
- Les probabilités à priori $p(\omega_i)$ pour $i = 0..9$. Elles s'obtiennent tout simplement par fréquence relative, cad. le nombre d'éléments d'une classe divisé par le nombre d'éléments au total.
- Les probabilités conditionnelles $P(x|\omega_i)$. On suppose que celles-ci puissent être approchées par des Gaussiennes de dimension 256. Il s'agit donc d'obtenir la moyenne μ_i (vecteur de dimension 256) et la matrice de covariance Σ_i (matrice de dimension 256×256), **séparément** pour chaque classe ω_i.

Rappel de l'équation d'une Gaussienne :

$$p(x) = \frac{1}{\sqrt{2 \cdot \pi} \|\Sigma\|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^t \Sigma^{-1} (\mathbf{x}-\mu)}$$

Une fois que toutes ces valeurs sont calculées, il convient de les stocker dans un fichier (`write` en MATLAB) afin qu'elles peuvent être utilisées dans la phase de classification.

2 Phase de classification

On traite séquentiellement tous les exemples du corpus de développement. Pour chaque exemple on calcule l'expression $P(\omega_i|x) = P(x|\omega_i)p(\omega_i)$ avec les paramètres déterminés lors de la phase d'apprentissage, et ceci pour toutes les 10 classes $i = 0..9$. La classe dont l'expression ci-dessus est la plus grande est la classe reconnue. Puisque vous connaissez la bonne classe pour chaque exemple du corpus de développement, vous pouvez ainsi compter le nombre d'erreurs.

Comparer les performances de plusieurs variantes de votre classifier.

3 Phase d'évaluation

Vous utiliserez votre meilleur système pour classer tous les exemples du corpus d'évaluation de la même manière que ci-dessus. Cependant, vous ne connaissez pas les bonnes réponses pour ces données, mais il faut envoyer par émail un fichier à votre professeur avec le numéro de la classe reconnue par ligne. Il calculera ensuite le taux d'erreur de votre système.

4 Réduction de la dimension

Le calcul d'une Gaussienne multi-dimensionnelle implique une inversion de la matrice de covariance Σ^{-1}. Lorsque vous faites ce calcul pour des données de dimension 256, vous obtiendrez une erreur. En effet, certains pixels des images 16x16 ne changent jamais de valeur (p.ex. les bords des images du chiffre « 1 »), leur variance est donc zéro, et il est impossible d'inverser la matrice.

Pour remédier à ce problème, nous utilisons une technique appelée « analyse en composantes principales (ACP) » qui permet de réduire la dimension des données. L'idée consiste à trouver une projection des données qui préserve le mieux la variabilité des données. Le code pour cette opération est fournie (fichier « acp.m » dans le répertoire des données). Vous pouvez librement choisir la dimension de la projection (essayer des valeurs entre 10 et 100). Faites plusieurs expériences pour déterminer la dimension de projection qui donne le taux d'erreur le plus faible. Tracer une figure qui montre lex taux d'erreurs en fonction de la dimension de la projection.

5 Tableau de confusions

Produire un tableau de confusions pour votre meilleur système. Il s'agit d'un tableau avec dix lignes et dix colonnes. Chaque ligne donne les erreurs de classification spécifique à un chiffre, p.ex. il y 10 chiffres « 1 » qui sont mal reconnus dont 8 sont pris pour des « 7 », et deux pour des « 4 ».

6 Quelques rappels sur MATLAB

MATLAB est un langage de programmation adapté au traitement de problèmes numériques. La syntaxe correspond beaucoup aux langages habituels (C++, Java, etc) : il y a des fonctions, des boucles, etc. Cependant, une variable peut directement contenir un vecteur ou une matrice et MATLAB connaît un grand nombre d'opérations sur des vecteurs et matrices. Il n'est pas nécessaire de déclarer les variables et un vecteur peut changer de dimension sans problème.

Quelques exemples :

```
A = [ 1 2 3; 4 5 6]
A(:,1)  % extraire la première colonne
i=[1 3];  % le point virgule supprime l'affichage immédiat du résultat
A(:,i) % extraire les colonnes dont les indices sont dans la variable i, ici 1re et 3e
x=1:3;
A*x'    % x' denote x transposé
Z=rand(3,3)
inv(Z) % inverse d'une matrice
det(Z) % déterminant d'une matrice

for i=1:10
 y(i)=i*i;
end
y

% plus efficace :
i=1:10
y=i .* i; % multiplication élement par élement

% la plupart des fonctions peuvent opérer sur des vecteurs
x=linspace(0,2*pi,1000);
y=sinx(x)
plot(x,y)

% trouver le nombre d'éléments supérieurs à 0,5
x=rand(1,1000);
i=find(x>0,5); % indices de tous les éléments dont x > 0,5

function y=carre(x)
 y=x*x;
endfunction

z=carre(3)
```

Il y a des fonctions prédéfinies pour beaucoup d'opérations courantes : fonctions exponentielles, logarithmiques, trigonométriques, etc, sommer des composantes d'un vecteur (`sum`), calculer la moyenne et la covariance , ... Chaque fonction est documentée, p.ex. **help sum**.