MA201 - MODÈLES PROBABILISTES ET STATISTIQUES - PROJET

ENSTA PARISTECH

Classification automatique de chiffres manuscrits par maximum de vraisemblance

Jean-Baptiste Bordes

21 septembre 2018

1 Introduction

Le but de ce projet est de faire de la reconnaissance automatique de caractères manuscrits en utilisant le critère de maximum de vraisemblance pour apprendre un modèle de mélange de gaussiennes. En faisant varier le nombre de gaussiennes présentes dans le mélange, il est possible d'adapter finement la complexité du modèle à la base de données considérée.

1.1 Description de la base de données MNIST

La base de données MNIST de caractères manuscrits contient 60000 images exemple d'apprentissage et 10000 images exemples de test. Ces images sont centrées et normalisées en taille à 28×28 pixels. Les caractères sont des chiffres manuscrits allant de 0 à 9. De nombreuses équipes de recherche ont testé leurs modèles sur cette base de données, vous pouvez voir les résultats obtenus et télécharger les bases de données sur le site http://yann.lecun.com/exdb/mnist/.

La base de données consiste en quatre fichiers:

- `train-images-idx3-ubyte` contient le jeu d'images d'apprentissage.
- `train-labels-idx1-ubyte` contient les annotations des images d'apprentissage (c'est à dire les chiffres correspondants à chaque image d'apprentissage)
- `t10k-images-idx3-ubyte` contient le jeu d'images de test
- `t10k-labels-idx1-ubyte` contient les annotations des images de test (c'est à dire les chiffres correspondants à chaque image d'apprentissage)

Le script Python qui vous est fourni pour ce projet `main.py` vous procure le code pour lire ces fichiers et les stocker dans des tableaux numpy.

Pour extraire des caractéristiques efficaces pour faire de la reconnaissance de caractères un vecteur SIFT est calculé au centre de chaque image[2]. Le SIFT est un vecteur de caractéristiques particulièrement efficace en traitement d'images. Il s'agit d'un vecteur de taille 128 qui décrit des propriété spatiales sur la base d'un décompte des gradients locaux. Les vecteurs SIFT ont déjà été calculés sur chaque image du jeu de données. Le code pour lire ce fichier et le stocker dans des tableaux est également déjà

implémenté dans `main.py`. La constante `size_features` qui est déjà défini au début de `main.py` est le nombre de composantes que nous gardons pour chacun des vecteurs. Initialement, nous le fixons à 8 pour éviter des calculs trop longs. Il est possible cependant de monter à 128.

1.2 Modèle de mélange de gaussiennes

Nous introduisons un modèle probabiliste pour décrire la distribution des caractéristiques en fonction du chiffre présent dans l'image par un mélange de gaussiennes. Formellement, trois variables aléatoires X, Y, Z sont introduites. X au vecteur de caractéristiques. Y est le chiffre correspondant (de 0 à 9). Z est une variable aléatoire supplémentaire "cachée" qui prend ses valeurs dans $\{1, \ldots, K\}$ où K est la complexité du modèle. Le rôle de cette variable supplémentaire est de prendre en compte différents types d'écriture d'un même caractère.

Un modèle de mélange de gaussienne est défini pour chaque couple (Y, Z):

$$P(X = x|Y = i, Z = j) = \frac{\exp(-(x - \mu_{ij})^t \Sigma_{ij}^{-1}(x - \mu_{ij}))}{\sqrt{(2\pi)^n |\Sigma_{ij}|}} = g_{ij}(x)$$

.

Dans ce projet, pour accélérer les calculs, la matrice de covariance des gaussiennes sera supposé diagonale.

La vraisemblance d'un vecteur x d'être annoté par un chiffre i est donc obtenu par la marginalisation sur Z:

$$P(X = x|Y = i) = \sum_{j=1}^{K} P(Z = j|Y = i) P(X = x|Y = i, Z = j)$$

.

Notons $P(Z = j|Y = i) = \alpha_{ij}$, qui peut être interprété par l'importance relative des gaussiennes dans le mélange:

$$P(X = x|Y = i) = \sum_{j=1}^{K} \alpha_{ij} g_{ij}(x)$$

Pour un chiffre donné i, un mélange de K gaussiennes a pour paramètres:

- K matrices de covariances Σ_{ij} of size $n \times n$. Comme dit plus haut, dans ce projet, les matrices sont supposées diagonales et on retient donc K vecteurs de taille n.
- K vecteurs de moyenne μ_{ij} de taille n.
- K coefficients de poids α_{ij}.

Dans le fichier `gmm.py`, la fonction `loggmm()` fournit la log-vraisemblance (l'opposé du logarithme de la probabilité) du modèle de mélange de gaussiennes pour un vecteur de caractéristiques et des paramètres donnés.

2 Création d'un module de reconnaissance automatique de chiffre

Nous allons implémenter le modèle de reconnaissance en utilisant un modèle de mélange de gaussiennes. Deux étapes doivent être implémentés: l'étape d'apprentissage et l'étape de classification.

2.1 Étape d'apprentissage

Pour apprendre le modèle, nous allons utiliser l'algorithme Expectation Maximization (EM), qui est particulièrement bien adapté pour apprendre des modèles comportant une variable cachée. Le principe est de calculer une boucle itérative de deux étapes: l'étape *E* qui tente de deviner la valeur de la variable Z pour chaque échantillon de l'apprentissage, et l'étape *M* qui effectue l'apprentissage du modèle sur la base des valeurs choisies pour Z.

Voici le pseudo-code pour chaque chiffre i dans $\{0, \ldots, 9\}$:

- Initialiser Σ_{ij}, μ_{ij} et α_{ij} pour $j \in \{1, \ldots, K\}$
- Jusqu'à convergence, calculer:
  - Étape E: pour chaque échantillon x de classe i dans la base de données d'apprentissage, trouver $j \in \{1, \ldots, K\}$ vérifiant $max(g_{ij}(x))$. Stocker cette valeur dans une matrice W.
  - Étape M: pour chaque $j \in \{1, \ldots, K\}$, estimer Σ_{ij} et μ_{ij} sur la base des échantillons x vérifiant $W(x) = j$.
  - Pour $j \in \{1, \ldots, K\}$, set $\alpha_{ij} = \frac{\#(W(x)==j)}{\#training\ samples\ of\ class\ i}$ (# correspond au nombre d'éléments)

2.2 Étape de classification

Étant donné un vecteur de test x, la classification va être faite en utilisant le critère de maximum de vraisemblance

$$C(x) = argmax_{i \in \{0,\ldots,9\}} (\sum_{j=1}^{K} \alpha_{ij} g_{ij}(x)),$$

L'idée sous-jacente à ce critère est que le chiffre correspondant à l'image est celui pour lequel la vraisemblance du vecteur de caractéristiques est la plus grande.
où $C(x)$ correspond au chiffre qui classifie un vecteur de caractéristiques x.

2.3 Pipeline du classificateur

Le pipeline du classificateur est dans le fichier `main.py`. Initialement, les paramètres sont définis et le tableau pour stocker la précision du module de reconnaissance est initialisé.

```
# parameters for the lab (can be changed for the bonus question)
size_features=8 # number of features retained from the PCA
size_training=1000; #number of samples retained for training
size_test = 1000; #number of samples retained for testing
K_max = 10 # maximum complexity of the mixture - number of GDs / digit (class)

#arrays to store results
results = zeros((K_max,2))
```

Les données sont ensuite chargées. Elles sont composées des images brutes, des chiffres correspondant et des caractéristiques calculées, pour le jeu d'apprentissage et de test.

```
#arrays to store results
results = zeros((K_max,2))
#reading of the dataset
images, labels_training, images_test, labels_test = read_dataset(size_training, size_test,
```

```
size_features);

#reading of the features extracted from dataset
features_test=numpy.array(list(csv.reader(open("test_data.csv","rb"),delimiter=',')))
        .astype('float') #loading the PCA features of the test data set
features_training=numpy.array(list(csv.reader(open("training_data.csv","rb"),delimiter=',')))
        .astype('float') #loading the PCA features of the training data set
features_test = features_test[:size_test,:size_features]
#only "size_features" first features are kept for training set
features_training = features_training[:size_training,:size_features]
     #only "size_features" first features are kept for test set
```

Ensuite, les paramètres du modèle sont initialisés. Nous avons stocké les paramètres des gaussiennes pour chaque chiffre et chaque composante du mélange, de même que les poids.

```
# arrays containing the model for the mixture
mean_mix = zeros((10,K_max,size_features)) # mean values for the Gaussian distributions
var_mix = zeros((10,K_max,size_features)) # variance for the Gaussian distributions
alpha_mix = zeros((10,K_max)) # mixture weights for the Gaussian distributions
```

2.4 Travail à effectuer

***Question 1*:** Compléter la fonction `mix_gmm_em()` dans le fichier `mix_gmm_em.py` qui effectue l'algorithme Expectation Maximisation. Cette fonction prend en entrée les données ainsi qu'un nombre K et renvoie les paramètres du modèle qui ont été estimés.

Écrire une étape d'initialisation de cet algorithme. Justifier le choix que vous avez effectué.

***Question 2*:** Remplir le code de la fonction `classify_gmm()` dans le fichier `classifier.py`. Cette fonction doit renvoyer le chiffre correspondant à un vecteur de caractéristiques calculées sur un eimage.

***Question 3*:** Faites varier K de 1 à 20 pour une taille de caractéristiques fixée. Représenter sur un même graphique le taux d'erreur sur:

- le jeu d'apprentissage
- le jeu de test

Commenter les résultats obtenus. Comment se caractérise le sur-apprentissage ?

Question 4 Faites varier K de 1 à 30 ainsi que la taille des caractéristiques de 5 à 50. Représenter le taux d'erreur sur le jeu de test par une surface 2D. Quel est le meilleur résultat que vous obtenez?

Question 5 On cherche à trouver automatique la valeur optimale du nombre de gaussiennes dans le mélange. Pour cela, on utilise un outil de théorie de l'information nommé "Minimum Description Length" (MDL) [3]. Il s'agit d'un critère qui stipule que, lorsque plusieurs modèles sont possibles, le meilleur est celui qui permet de coder la base de données par un nombre de bits minimum. Il est alors nécessaire de coder le modèle qui permet de représenter la base de données, et la base de données à l'aide du modèle: Le nombre total de bits $C(X, M)$ s'écrit alors:

$$C(X, M) = C(M) + C(X|M)$$

$C(M)$ est le nombre de bits nécessaire pour coder le modèle. Il s'agit de coder tous les paramètres du modèle. Plus le modèle est complexe, et plus cette quantité augmente. $C(X|M)$ est le nombre de bits nécessaire pour coder les données sachant le modèle. Plus le modèle est sophistiqué, plus cette quantité sera faible. Le MDL fournit ainsi un outil permettant de rechercher un compromis entre complexité du modèle et qualité de la représentation des données.

Pour exprimer ces termes, on utilisera les formules suivantes:

- $C(M)$: Un paramètre étant un nombre réel, il faudrait en principe utiliser un nombre infini de bits pour le coder. Cependant, les paramètres étant estimés avec un nombre fini d'échantillon, leur précision est limitée. Rissanen propose la quantité $\frac{n}{2}log(N)$ bits pour coder un vecteur de paramètres de n échantillons estimé avec N échantillons.
- $C(X|M) = log(p_M(X))$, où $p_M(X)$ est la probabilité de la base de données X estimée par le modèle M.

Calculer $C(X|M]$ en faisant varier K de 1 à 30 ainsi que la taille des caractéristiques de 5 à 50. Tracer sur une surface 2D le résultat obtenu: quel est le minimum obtenu ? Cela coincide-t-il avec le meilleur résultat obtenu sur la base de test ? Commenter.

References

[1] Yann LeCun, Corina Cortest, Christopher J.C. Burges. "The MNIST database of handwritten digits". `http://yann.lecun.com/exdb/mnist/` Dash, Manoranjan, and Huan Liu. "Feature selection for classification". Intelligent data analysis 1.3 (1997): 131-156.

[2] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision (2004): 91-110

[3] Rissanen, J. (1978). "Modeling by shortest data description". Automatica. 14 (5): 465–658