Master 2 ATAL

Signal et Langue

Digit Recognizer

Étudiants :
Coraline Marie
Romain Rincé

Encadrants :
Harold Mouchère
Christian Viard-Gaudin

15 novembre 2014

UNIVERSITÉ DE NANTES

Table des matières

Introduction

1 Hidden Markov Models

1.1 Extraction de séquences

Pour cette première partie, seul quelques extraits de codes seront présentés. L'intégralité du code source se trouve dans le fichier `src/hmmPart1.r` du projet.

Extension de la fonction `simu_symbol`

La fonction `simu_symbol` permet de construire et d'afficher des chiffres. Pour cela, elle récupère un ensemble de 30 points différents, qu'elle relie ensuite dans un odre précis. L'extension de cette fonction par le code ci-dessous, permet désormais de construire le chiffre 6.

```
simu_symbol <- function()
{
  (...)
  digit_6 <- rbind(
    stroke(-0.5,1.0,-0.5,-0.5,10),
    stroke(-0.4,-0.5,0.5,-0.5,7),
    stroke(0.5,-0.4,0.5,0.4,6),
    stroke(0.4,0.4,-0.5,0.4,7)
  )
  dimnames(digit_6) <- list(num=1:nrow(digit_6),point=c("x","y"))
  plot(digit_6,type="l",col="red",xlim=c(-1,1),ylim=c(-1,1))
  points(digit_6)
  return(list(d1=digit_1,d2=digit_4,d3=digit_6))
}
```

Analyse de la fonction `compute_symbol`

La fonction `compute_symbol` prend trois paramêtres en entrée : le tracé d'un chiffre, un nombre de lignes (5) et un nombre de colonnes (3). Elle commence par construire une matrice, ayant pour dimmension le nombre de lignes et de colonnes précisés précédemment. Cette matrice servira à positionner les points du tracé analysé :

$$\begin{pmatrix} 13 & 14 & 15 \\ 10 & 11 & 12 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

Elle calcule ensuite le nombre de points présents dans le tracé du chiffre, puis les positions horizontales et verticales de chacun de ces points. Pour finir, elle détermine la position de chaque point dans la matrice qu'elle a créée, et les affiche.

Voici les resultats donnés par cette fonction pour les trois tracés créés par la fonction `simu_symbol` :

```
pour 1 : 11 11 14 14 14 14 14 14 14 14 14 14 14 14 11 11 11 11 8
    8 8 8 5 5 5 5 2 2 2 2
pour 4 : 14 14 14 11 11 10 7 7 7 4 4 4 4 4 5 8 8 8 9 9 8 8 8 5 5
    5 2 2 2 2
pour 6 : 13 13 13 10 10 7 7 7 4 4 4 5 5 5 5 6 6 6 6 9 9 12 12 12
    11 11 11 11 10 10
```

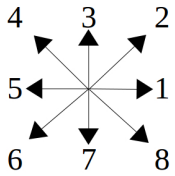
Ce qui donne pour le chiffre "6", les positions suivantes :

$$\begin{pmatrix} 13 & 14 & 15 \\ 10 & 11 & 12 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

Avec une matrice contenant suffisamment de lignes et de colonnes, cette méthode permet de déterminer efficacement la position de chaque point d'un caractère. Cependant, si le caractère analysé est mal positionné (décalage à gauche ou à droite), toutes les positions sont également décalées au niveau de la matrice. Il faut donc faire un gros prétraitement pour que le caractère soit bien découpé et bien positionné. De plus, l'ordre de chaque points est essentiel pour la construction de la séquence.

Analyse de la fonction `compute_symbol_dir`

La fonction `compute_symbol_dir` prend deux paramètres en entrée : le tracé d'un chiffre et un nombre d'angles qui serviront de direction (ici 8). Chaque direction est numérotées telles que :

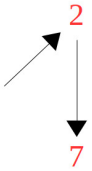


Ainsi, la direction 1 indique un déplacement vers la droite, la direction 5 vers la gauche, etc...

Voici les resultats donnés par cette fonction pour les trois tracés créés par la fonction `simu_symbol_dir` :

```
1  pour 1 : 2 2 2 2 2 2 2 2 2 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
       7 7
2  pour 4 : 6 6 6 6 6 6 6 6 6 5 1 1 1 1 1 1 1 1 1 4 7 7 7 7 7 7 7 7
       7 7
3  pour 6 : 7 7 7 7 7 7 7 7 7 1 1 1 1 1 1 1 3 3 3 3 3 3 5 5 5 5 5 5
       5 5
```

Ce qui donne pour le chiffre "1" deux tracés : le premier allant vers le 2 et le second allant vers le 7.

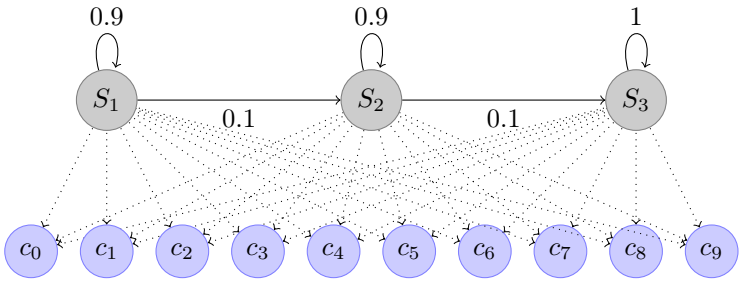


L'un des principaux avantages de cette méthode est que la taille de la séquence, ainsi que sa position (avec ou sans décalage) n'affectent pas les résultats.

Cette fonction arrive à déterminer précisément l'angle de la direction du tracé. Cependant, si l'image n'est pas dans le bon sens ou si l'écriture est inclinée ou en italique, la reconnaissance de l'angle sera faussée. De plus, le chiffre "1" et le chiffre "7" peuvent être facilement confondu à cause de certaines polices d'écriture. L'ordre de chaque points est également essentiel pour la construction de la séquence.

1.2 Création et initialisation d'un HMM

Pour cette nouvelle partie, nous considérerons un HMM à 3 états (S_x), observant 10 chiffres (c_x) et une topologie gauche-droite :



Pour ce HMM obervant une topologie gauche-droite, il est nécessaire que l'état S_1 soit l'état initial. Pour cela, les probabilités initiales attribuées aux états sont de 1 pour S_1, et 0 pour S_2 et S_3.

En ce qui concerne la matrice de transition, nous savons que la séquence T=30, et qu'il y a 3 états. Cela signifie qu'il n'y a qu'une chance sur dix de passer à l'état suivant et neuf chances sur dix de rester dans cet état. La matrice de transition est donc :

$$\begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

Sans plus d'informations sur les chiffres, la matrice d'observation sera initialisée avec une distribution uniforme.

1.3 Entraînement des modèles

Pour cette troisième partie, seul quelques extraits de codes seront présentés. L'intégralité du code source se trouve dans le fichier `src/hmmDigitR.r` du projet.

Création de la fonction `initHMMDigit`

La fonction `initHMMDigit` prend en entrée le nombre d'états et le nombre d'observations pour le HMM. Elle initialise ensuite les probabilités initiales (1, 0, 0, ...), ainsi que les matrices de transitions et d'emissions.

```
1   initHMMDigit <- function(nbrEtats, nbrObsv){
2     (...)
3
4     # Mise en place des proba de transition pour le HMM optimal
5     (...)
6     matriceInit <- matrix(transProbs, nrow = nbrEtats, ncol =
          nbrEtats, byrow = T)
7
8     # Mise en place des proba d'emission
9     for (i in 1:nbrEtats*nbrObsv){
10    emissionProbs <- c(emissionProbs, 1/nbrObsv)
11    }
12    matriceEmission <- matrix(emissionProbs,nrow = nbrEtats, ncol =
           nbrObsv, byrow = T)
13
14    # Appel de initHMM
15    resultat <- initHMM(etats, obsv, probInit, matriceInit,
          matriceEmission)
16    return (resultat)
17  }
```

Une fois initialisés, les HMMs sont ensuite entraînés. Pour cela nous avons utilisé différents paramêtres :

— Train_compute_symbol (5x3, 5x4, dir_8 et dir_16)
— Le nombre d'états (3, 4 et 7)

Pour plus de simplicité, les models entrainés ont été sauvegardés dans le dossier `Results`. Ainsi, il est plus facile de les recharger. Les Hmms pour 7 états n'ont pas pu être entrainés pour tous les Train_compute_symbol, car cela était trop coûteux pour la machine.

1.4 Performances de la reconnaissance

Après la phase d'entrainement des HMMs, et avant de tester le système, il faut au préalable définir des models en réunissant les HMMs entrainés de même catégories. Chaque model est composé de 10 HMMs, un pour chaque chiffre ayant été entrainé par le même type de données (5x3, 5x4, ...).

Pour plus de facilité, et pour améliorer le temps de chargement, l'intégralité de ces modèles a été au pralable sauvegardé dans le dossier `Results`. Ainsi il est plus facile de les chargers et de lancer les tests.

La phase de test permet de mesurer grace aux fonctions :

le taux de reconnaissance de chaque HMM. De plus, toujurs dans l'optique d'améliorer la lecture des résultats

2 Réseaux de neurones

Conclusion