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, seulS quelques extraits de code 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 paramètres 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 dimension 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 résultats 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
```

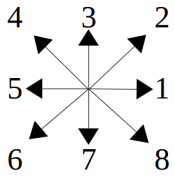
Ce qui donne pour le chiffre "6", les positions suivantes :

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

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 traitement important en amont pour que le caractère soit bien découpé et bien positionné. De plus, l'ordre de chaque point 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 numérotée telle que :

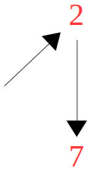


Ainsi, la direction 1 indique un déplacement vers la droite, la direction 5 vers la gauche, etc...

Voici les résultats 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.

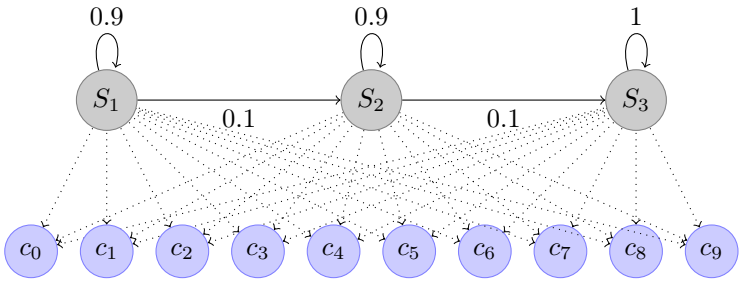


L'un des principaux avantages de cette méthode est que la taille de la séquence, ainsi que sa position (avec ou sans décalage) n'affectent pas les résultats.

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 confondus à cause de certaines polices d'écriture. L'ordre de chaque point 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 observant 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, seuls quelques extraits de code 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'émissions.

```
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 paramètres :

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

Pour plus de simplicité, les modèles 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'entraînement des HMMs, et avant de tester le système, il faut au préalable définir des models en réunissant les HMMs entraînés de même catégories. Chaque modèle est composé de 10 HMMs, un pour chaque chiffre ayant été entraîné 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 préalable sauvegardé dans le dossier `Results`. Ainsi il est plus facile de les charger et de lancer les tests.

La phase de test permet de mesurer, grâce aux fonctions `classify` et `recognition`, le taux de reconnaissance de chaque HMM. De plus, toujours dans l'optique d'améliorer la lecture des résultats

Résultats

Voici les résultats obtenus pour 3 états, avec les quatre types de données suivants :

Pour 5x3 : 81.5%

1		0	1	2	3	4	5	6	7	8	9
2	0	121	1	0	0	0	2	1	0	4	0
3	1	1	104	23	0	1	0	0	3	1	3
4	2	4	1	102	1	8	0	4	2	1	15
5	3	0	0	2	128	1	0	0	3	0	10
6	4	1	3	13	11	104	1	9	4	0	5
7	5	1	3	1	33	0	72	8	1	14	4
8	6	2	0	0	1	3	0	135	1	1	0
9	7	0	0	0	0	11	0	2	111	3	0
10	8	2	0	0	0	1	0	0	2	123	0
11	9	0	0	8	8	0	0	0	1	2	110

Pour 5x4 : 82.6%

1		0	1	2	3	4	5	6	7	8	9
2	0	117	3	0	0	0	2	0	0	7	0
3	1	3	105	9	3	7	0	1	0	0	8
4	2	3	1	109	3	4	0	1	4	1	12
5	3	0	0	1	130	1	0	0	2	0	10
6	4	0	2	2	9	122	0	7	3	0	6
7	5	3	5	0	35	0	59	9	1	23	2
8	6	5	0	0	0	1	0	135	0	2	0
9	7	0	0	1	0	14	0	0	110	1	1
10	8	2	1	0	0	1	1	0	2	121	0
11	9	0	0	2	8	0	1	0	1	0	117

Pour dir_8 : 81.4%

1		0	1	2	3	4	5	6	7	8	9
2	0	116	0	1	0	1	2	6	0	1	2
3	1	0	100	10	0	7	0	4	15	0	0
4	2	3	0	123	1	1	0	0	3	5	2
5	3	0	0	0	119	0	17	0	0	7	1
6	4	2	0	0	0	142	0	2	0	0	5
7	5	1	0	1	26	0	79	0	0	12	18
8	6	24	0	0	0	0	1	115	0	0	3
9	7	0	0	12	1	0	0	0	111	2	1
10	8	13	0	3	2	0	2	5	13	87	3
11	9	0	2	2	1	2	3	0	0	3	116

Pour dir_16 : 82.8%

1		0	1	2	3	4	5	6	7	8	9
2	0	115	1	1	1	1	1	6	0	1	2
3	1	0	107	13	0	5	0	1	7	2	1
4	2	1	1	123	0	0	0	0	3	4	6
5	3	0	0	0	125	0	9	0	0	8	2
6	4	0	0	0	0	144	1	1	0	1	4
7	5	1	0	0	30	0	89	4	4	6	3
8	6	15	0	0	0	0	1	122	0	1	4
9	7	0	0	5	1	0	1	1	116	3	0
10	8	6	1	4	1	0	2	7	16	82	9
11	9	2	1	3	2	6	4	0	0	6	105

2 Réseaux de neurones

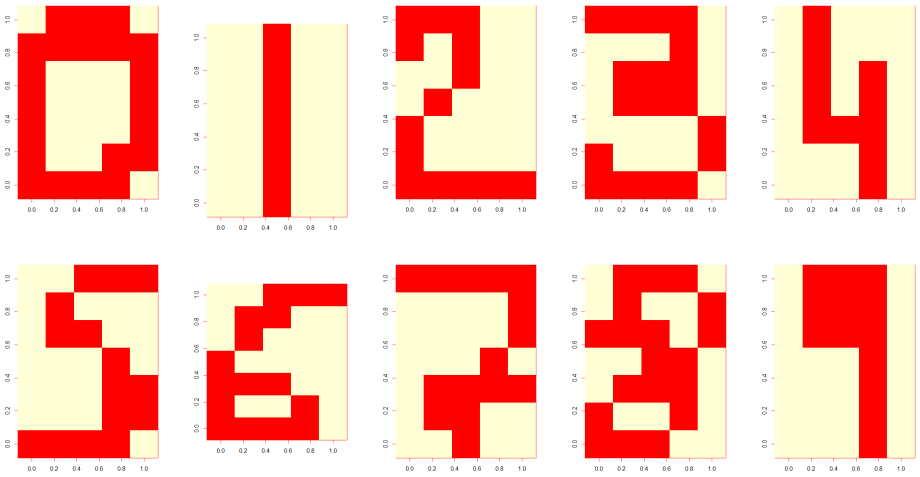
2.1 Extraction des caractéristiques

Afin de pouvoir utiliser un réseau de neurones pour apprendre à reconnaitre les chiffres, il était nécessaire au préalable d'extraire des caractéristiques des données de tracés fournies.

2.1.1 Données positionnelles

Pour les données positionnelles [1] trois types de caractéristique ont été extraites.

L'image du tracé L'image de tracé est reconstruite en créant une matrice de la taille de la grille des données d'entrées et de compter le nombre de fois qu'un point apparait à une coordonnée. Chaque cellule peut être considérée comme un pixel plus ou moins « lumineux ». Pour le réseaux de neurones la matrice est cependant fournit sous forme de vecteur.



Les projections sur les axes Une fois l'image reconstruite une projection sur l'axe X et Y sont effectuées, ce qui fournit deux vecteurs de caractéristiques supplémentaires.

Couverture de l'image Sur chaque début et fin de ligne ou colonne des sondes récupèrent le premier pixel allumé à partir du bord où elles sont parties. L'ensemble de ces informations sont concaténées dans un vecteur et fournies au réseau de neurones.

2.1.2 Données vectorielles

Ces données ont été traitées peu de temps et seulement deux vecteurs de caractéristiques ont été extraits.

1. Les coordonnées correspondent à des grilles de 5×3, 5×4 et 7×5,

L'occurrence des orientations de tracé À l'instar de l'image du tracé, le nombre des différentes orientations prises lors du tracés est mémorisé et stocké dans un vecteur [2].

L'occurrence des variations Il s'agit d'un vecteur qui contient le nombre de fois que le tracé a changé de direction et avec quel importance. Par exemple un « 1 » aura souvent 0 ° de variation tandis qu'un « 6 » aura souvent des variations du tracé entre 10 ° et 20 ° .

Les caractéristiques extraites sont ensuite données au réseau de neurones pour l'entrainement.

2.2 Méthodologie

2.3 Résultats

2. Par exemple : trois fois vers le bas, une fois à gauche...

Conclusion