| | | |
|---|---|---|
| | *Travaux Pratiques* | |
| | **Modélisation et analyse des systèmes**<br>*Modèles de Markov Cachés* | |

Rediger par:

TSAGUA YEMEWA Beyonce

I2FE

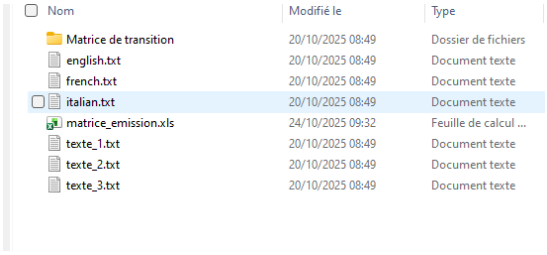
# RAPPORT TP(HMM)

Modélisation système

# Table des matières

# Etape 1: preparation des données

## Fichiers fournis



| Nom | Modifié le | Type |
|---|---|---|
| Matrice de transition | 20/10/2025 08:49 | Dossier de fichiers |
| english.txt | 20/10/2025 08:49 | Document texte |
| french.txt | 20/10/2025 08:49 | Document texte |
| italian.txt | 20/10/2025 08:49 | Document texte |
| matrice_emission.xls | 24/10/2025 09:32 | Feuille de calcul ... |
| texte_1.txt | 20/10/2025 08:49 | Document texte |
| texte_2.txt | 20/10/2025 08:49 | Document texte |
| texte_3.txt | 20/10/2025 08:49 | Document texte |

*Figure 1: Gestionnaire de fichiers_aborescence du dossier de travail fournir*

## Collecte et nettoyage des donnees

```python
"Lecture et nettoyage des corpus"

import unicodedata
import re

def lire_corpus(nom_fichier):
    texte_nettoye = ""
    try:
        with open(nom_fichier, 'r', encoding='utf-8') as f:
            for ligne in f:
                # Supprimer les accents
                ligne = unicodedata.normalize('NFD', ligne)
                ligne = ligne.encode('ascii', 'ignore').decode('utf-8')
                ligne = ligne.lower()
                # Supprimer la ponctuation et les caractères non a-z
                ligne = re.sub(r'[^a-z\s]', '', ligne)
                # Ajouter la ligne nettoyée au texte global
                texte_nettoye += ligne.strip() + " "
    except FileNotFoundError:
        print(f"Fichier '{nom_fichier}' introuvable.")
    except Exception as e:
        print(f"Erreur lors de la lecture du fichier : {e}")

    # Supprimer les espaces superflus et retourner sous forme de liste contenant
    return texte_nettoye.strip()
```

*Figure 2: fonction lire_corpus*

```python
corpus_fr = lire_corpus("C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/sem7/Modelisation Sys/TP/TP/fi
print(corpus_fr)
```

*Figure 3:test de la fonction lire_corpus*

```
Console 1/A ×
pretendu enfant du portier etait donc le sien et pourquoi ne le lui avait on pas dit tout de
suite j aurais du le deviner au reste ajouta t il son poupard est deja aussi laid et aussi camus
que lui tout ce denigrement superbe impatientait eugenie jusqu a l indignation elle cassa deux
assiettes et je crois que malgre sa douceur et la dignite habituelle de ses manieres elle eut
grande envie de jeter la troisieme a la tete d horace je la soulageai infiniment en prenant le
parti de dire tout de suite la verite puisque aussi bien horace devait l apprendre tot ou tard
il valait mieux qu il l apprit de nous et dans un moment ou nous pouvions en surveiller l effet
sur lui arsene m avait autorise depuis plusieurs jours et pour son compte et de la part de
marthe a agir comme je le jugerais utile en cette circonstance comment se fait il horace lui dis
je que vous n ayez pas devine deja que la femme de paul arsene est une personne tres connue de
vous et qui nous est infiniment chere il reflechit une minute en nous regardant alternativement
avec des yeux troubles puis prenant tout a coup une attitude degagee imitee du marquis de vernes
au fait dit il ce ne peut etre qu elle et je suis un grand sot de n avoir pas compris pourquoi
vous etiez si embarrasses tout a l heure devant la vieille fee qui emportait l enfant mais l
```

*Figure 4:Extrait du resultat obteenu*

# Construction du modele HMM ()

## Rappel

Un modèle de Markov caché modélise une séquence d'observations (dans notre cas, les lettres d'un mot) en fonction d'état caché. Un HMM ou automate de Markov à états cachés noté λ, est entièrement défini par le triplet de matrices noté : $\lambda = (\pi, A, B)$.

La structure de votre modèle de Markov caché sera définie par :

- Un ensemble d'états cachés $\boldsymbol{S} = \{S_1, S_2, \ldots S_N\}$. Où $S_i$ est état $i$ et $N$ le nombre d'états cachés du modèle.
- Le vecteur de probabilité initiale $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$. Où $\pi_i$ est la probabilité que $S_i$ soit l'état initial.
- La matrice de transition $\boldsymbol{A} = (a_{ij})$, dont les éléments $a_{ij}$ sont les probabilités de transition d'un état $S_i$ vers un état $S_j$,
- La matrice d'observation (ou d'émission) $\boldsymbol{B} = (b_i(O_k))$, dont les éléments $b_i(O_k)$ sont les probabilités d'émettre le symbole $O_k$ par l'état $S_i$.

## Gestion des matrices

### Création de la matrice d'émission

```python
def matrice_emission(nom_fichier_emission):
    # Lecture du fichier Excel
    B = pd.read_excel(nom_fichier_emission,index_col=0)
    return B.to_numpy()

print(matrice_emission("C:/Users/GLC/Desktop/3IL/Semestre7/I2_
```

*Figure 5:Fonction de la matrice emission*

```
In [2]: runfile('C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/sem7/Modelisatio
Chaine_Markov_Cachées.py', wdir='C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/
TP')
[[0.57142857 0.         0.08571429 0.         0.08571429 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.11428571 0.         0.         0.
  0.02857143 0.         0.05714286 0.05714286 0.         0.
  0.         0.        ]
 [0.         0.68965517 0.         0.06896552 0.         0.03448276
  0.         0.06896552 0.         0.         0.06896552 0.06896552
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.05882353 0.         0.58823529 0.         0.08823529 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.05882353 0.         0.         0.
  0.02941176 0.         0.08823529 0.08823529 0.         0.
  0.         0.        ]
 [0.         0.07407407 0.         0.74074074 0.         0.07407407
  0.         0.03703704 0.         0.         0.03703704 0.03703704
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

*Figure 6:Apercue du resultat obtenu*

## Creation de la matrice de transition

```python
def matrice_transition(nom_fichier):
    # Extraire toutes les lettres du corpus
    X = lire_corpus(nom_fichier)
    n = 26
    M = np.zeros((n, n))  # matrice 26x26

    # Remplir la matrice de transition
    for i in range(len(X) - 1):
        # Convertir les caracteres en indices numeriques
        c1 = ord(X[i]) - ord('a')
        c2 = ord(X[i+1]) - ord('a')
        # Eviter les espaces entre les caracteres
        if 0<= c1 <=25 and  0<= c2 <=25:
            M[c1,c2] += 1

    #matrice stochastique
    ligne = M.sum(axis=1)

    # eviter les division par 0
    ligne[ligne==0] = 1

    A = M/ligne[:,np.newaxis]


    return A
```

*Figure 7:Fonction matrice de transition*

```
In [3]: runfile('C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/sem7/Moc
Chaine_Markov_Cachées.py', wdir='C:/Users/GLC/Desktop/3IL/Semestre7/
TP')
Matrice transition fichier french.txt
 [[0.00000000e+00 2.10692384e-02 4.61334894e-02 9.64401130e-03
  7.63378203e-05 6.51416067e-03 2.38428459e-02 2.21379679e-03
  2.68251101e-01 1.17051324e-03 7.63378203e-05 3.60568971e-02
  3.56497621e-02 1.53311789e-01 1.52675641e-04 1.83974147e-02
  2.62093183e-03 1.06949286e-01 6.81187816e-02 4.14259905e-02
  7.72029823e-02 7.58543474e-02 0.00000000e+00 1.52675641e-04
  5.01285020e-03 1.01783760e-04]
 [8.17595578e-02 0.00000000e+00 0.00000000e+00 0.00000000e+00
  1.32657761e-01 0.00000000e+00 0.00000000e+00 2.30308614e-04
  2.11883924e-01 9.21234454e-03 0.00000000e+00 2.66236757e-01
  0.00000000e+00 6.90925841e-04 1.29894058e-01 0.00000000e+00
  0.00000000e+00 1.04329802e-01 3.43159834e-02 2.53339475e-03
  2.48733303e-02 2.30308614e-04 0.00000000e+00 0.00000000e+00
  1.15154307e-03 0.00000000e+00]
```

*Figure 8:Apercue du resultat*

- La somme de chacune des lignes de cette matrice vaut 1.

# Detection de la langue

## Probleme de premier type

### Code de la fonction forward

```python
def forward(O, A, B, PI):
    N = len(A) #Nombre d'etat
    T = len(O) #longueur de la sequence d'observation
    alpha = np.zeros((T,N))

    for i in range (N):
        alpha[0,i] = PI[i,0] * B[i,O[0]]

    for t in range(T-1):
        for j in range(N):
            somme = 0
            for i in range(N):
                somme += alpha[t,i]*A[i,j]
            alpha[t+1,j] = B[j,O[t+1]]*somme

    P = 0
    for i in range(N):
            P += alpha[T-1,i]

    return P, alpha
```

*Figure 9:Fonction forward*

## Code de la fonction backward

```python
def backward(O, A, B, PI):
    N = len(A)
    T = len(O)
    beta = np.zeros((T, N))
    # Initialisation
    beta[T-1, :] = 1
    # Boucle d'induction
    for t in range(T-2, -1, -1):
        for i in range(N):
            beta[t, i] = np.sum(A[i, :] * B[:, O[t+1]] * beta[t+1, :])
    # Terminaison
    P = np.sum(PI * B[:, O[0]] * beta[0, :])
    return P, beta
```

*Figure 10:Fonction backward*

i) Evaluation du modele

```python
def mot_en_indices(mot):
    return [ord(c)-ord('a') for c in mot.lower() if c.isalpha()]
```

*Figure 11: Fonction necessaire pour appliquer forward et backward lors des tests*

```python
mots  = ['probablement', 'probably','probabilmente' ]
O = mot_en_indices('probablement')

P_fr, alpha = forward(O,A_fr, B, PI)
P_en, alpha = forward(O,A_en, B, PI)
P_it, alpha = forward(O,A_it, B, PI)

P_fr_b, alpha = backward(O,A_fr, B, PI)
P_en_b, alpha = backward(O,A_en, B, PI)
P_it_b, alpha = backward(O,A_it, B, PI)

probs = np.array([P_fr, P_en, P_it])  # tes P(O|λ)
priors = np.array([1/3, 1/3, 1/3])   # probas a priori (égales ici)
posterior = probs * priors
posterior /= posterior.sum()  # normalisation

print("Probabilite du mot Probablement avec la fonction forward\n les resultats

probs = np.array([P_fr_b, P_en_b, P_it_b])  # tes P(O|λ)
posterior_b = probs * priors
posterior_b /= posterior_b.sum()  # normalisation

print("Probabilite du mot Probablement avec la fonction backward\n les resultat
```

*Figure 12: Code de tests des fonctions*

```
In [14]: runfile('C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/sem7/Modelisation
Chaine_Markov_Cachées.py', wdir='C:/Users/GLC/Desktop/3IL/Semestre7/I2_2026/sem
TP')

Probabilité du mot 'probablement' avec la fonction forward
Les résultats obtenus furent normalisés :
FR = 0.7696817064597231, EN = 0.05140610564574384, IT = 0.17891218789453306

Probabilité du mot 'probablement' avec la fonction backward
Les résultats obtenus furent normalisés :
FR = 0.7696817064597231, EN = 0.051406105645743876, IT = 0.178912187894533

----------------------------------------------------------------------

Probabilité du mot 'probably' avec la fonction forward
Les résultats obtenus furent normalisés :
FR = 0.04092028508338799, EN = 0.9503528361920396, IT = 0.008726878724572411

Probabilité du mot 'probably' avec la fonction backward
Les résultats obtenus furent normalisés :
FR = 0.04092028508338799, EN = 0.9503528361920396, IT = 0.008726878724572413

----------------------------------------------------------------------

Probabilité du mot 'probabilmente' avec la fonction forward
Les résultats obtenus furent normalisés :
FR = 0.29308659269592924, EN = 0.050398550305981095, IT = 0.6565148569980898

Probabilité du mot 'probabilmente' avec la fonction backward
Les résultats obtenus furent normalisés :
FR = 0.29308659269592935, EN = 0.0503985503059811, IT = 0.6565148569980895

----------------------------------------------------------------------
```

*Figure 13: Résultats obtenues de ces deux fonctions(forward et backward) _fonction d'affichage améliorer avec ChatGpt*

Nous pouvons donc conclure que probablement appartient a la langue francaise, le mot probably appartient à la langue anglaise et que celui probablimente appartient a la langue Italienne.

## Matrice de confusion

```python
def tester_et_confusion(mots, A_fr, A_en, A_it, B, PI, mot2idx, forward, backward):
    langues = ["FR", "EN", "IT"]
    matrices = [A_fr, A_en, A_it]
    priors = np.array([1/3, 1/3, 1/3])

    # matrice de confusion 3x3 initialisée à 0
    confusion = np.zeros((3, 3), dtype=int)

    for i, mot in enumerate(mots):
        # convertir mot → indices
        O = mot2idx(mot)

        # forward
        P_fw = np.array([forward(O, A, B, PI)[0] for A in matrices])
        posterior_fw = P_fw * priors
        posterior_fw /= posterior_fw.sum()

        # backward
        P_bw = np.array([backward(O, A, B, PI)[0] for A in matrices])
        posterior_bw = P_bw * priors
        posterior_bw /= posterior_bw.sum()

        # prédiction = langue avec la plus grande probabilité (forward ici)
        predicted = np.argmax(posterior_fw)

        # vérité terrain (FR=0, EN=1, IT=2)
        true_class = i

        # mise à jour de la matrice de confusion
        confusion[true_class, predicted] += 1

        print("\n------ Résultats pour le mot :", mot, "------")
        print("Posterior FORWARD  :", posterior_fw)
        print("Posterior BACKWARD :", posterior_bw)
        print("Langue prédite     :", langues[predicted])

    return confusion
```

*Figure 14: Code de test de la langue et de la matrice de confusion*

```
===== MATRICE DE CONFUSION =====
[[1 0 0]
 [0 1 0]
 [0 0 1]]

In [17]:
```

*Figure 15:Resultat obtenu*

Nous pouvons donc constater une parfaite classification car nous obtenons une matrice identite.

# Reconnaisance de la langue d’un texte

## Fonction pour lire le fichier texte

```python
def lire_fichier(nom_fichier):

    O = []  # liste des mots extraits

    with open(nom_fichier, 'r', encoding='utf-8') as f:
        for ligne in f:
            # Passage en minuscules
            ligne = ligne.lower()

            # Extraction de tous les mots (lettres uniquement)
            mots = re.findall(r"[a-zàâçéèêëîïôûùüÿñæ]+", ligne)

            # Ajout dans la liste principale
            O.extend(mots)

    return O
```

*Figure 16: Code de la fonction lire_fichier*


*Figure 17: Resultat de la fonction lire_fichier*

## Déduisons en la matrice de confusion

```
Matrice de confusion (valeurs normalisées 0-1) :
      FR    EN    IT
FR  0.66  0.15  0.19
EN  0.15  0.68  0.17
IT  0.25  0.23  0.51
```

*Figure 18: Matrice de confusion*

## Discussions sur la classification des différents mots

La classification correcte ou incorrect des différents mots qui constitue ces fichiers de tests dépendent en grande partie de la longue de chacun des mots du texte, de leurs structures internes (suffixe, préfixe, racine) et ainsi que de la fréquence de chacune des lettres dans les mots.

## Evaluation de l'influence du nombre de lettre qui compose un mot et conclusion

En effet, la longueur des observations a une influence sur la prédiction de la langue de ce mot. En général, les mots courts (03 caractères) sont très souvent mal classes, les mots moyen (05 à 07 caractères) ont de faibles changes d’être mal classée par rapport au mot court tandis que ceux longs (08 caractères et plus) ont tendance à être plutôt très bien classes.

# Remplaçons la matrice d'émission par la matrice Identité

## Matrice de Confusion obtenu

```
Matrice de confusion (valeurs normalisées 0-1) :
      FR    EN    IT
FR  0.94  0.02  0.04
EN  0.01  0.97  0.02
IT  0.23  0.19  0.59
```

## Discussions sur la classification des différents mots

**Bien classés :**

- Mots contenant des bigrammes ou trigrammes typiques (ex. "th", "wh", "gli", "sch", "qu")
- Mots longs (10+ lettres),
- Mots très typiques du corpus d'entraînement.

**Mal classés :**

- Mots courts,
- Mots fréquents dans plusieurs langues,
- Mots avec structure générique (ex. "son", "man", "uno", "non", "ton", etc.),
- Mots dont les transitions internes peuvent exister dans plusieurs langues.

## Evaluation de l'influence du nombre de lettre qui compose un mot et conclusion

**Mots courts**

→ quasi impossible à classer correctement (performance très faible)

**Mots moyens**

→ résultats très aléatoires
→ dépend trop des transitions internes, parfois ambiguës

**Mots longs**

→ meilleure classification
→ mais toujours moins bons que lorsque la matrice d'émission est utilisée

# Conclusion sur l'impact de la matrice d'émission

La matrice d'émission a une influence déterminante sur la qualité du modèle. Elle introduit l'information essentielle sur les fréquences des lettres propres à chaque langue, ce qui permet de distinguer efficacement les trois langues même lorsque leurs transitions sont semblables.
Lorsque l'on remplace la matrice d'émission par l'identité, les performances chutent fortement : les scores deviennent proches, la matrice de confusion perd sa structure, et la classification devient largement tributaire de la longueur des mots.
Ainsi, la matrice d'émission est indispensable pour obtenir un modèle linguistique performant. Sans elle, le HMM perd la majorité de sa capacité discriminante.