SI 221-B

Bases de l'Apprentissage

TP "Partie Chaine de Markov (Part 1)"

MARTINS BRAGA Arthur

In [ ]:
```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

filename_A='bigramenglish.txt'
A=np.loadtxt(filename_A)
print(len(A))
filename_F='bigramfrancais.txt'
F=np.loadtxt(filename_F)
print(len(F))
dic={'1' : ' ', '2' : 'a', '3' : 'b', '4': 'c', '5' : 'd', '6' : 'e', '7': 'f', '8' : 'g', '9' : 'h', '10': 'i', '11': 'j',
'12' : 'k', '13' : 'l', '14': 'm', '15' : 'n', '16' : 'o', '17': 'p', '18' : 'q', '19' : 'r' , '20': 's', '21' : 't', '22'
: 'u', '23': 'v', '24' : 'w', '25' : 'x' , '26': 'y', '27' : 'z', '28' : '.'}
print(len(dic))
```

```
28
28
28
```

I.2.a

bigramenglish.txt' contient la matrice des transitions pour l'anglais (bigrams) entre deux symboles (caractères ou espaces). Le terme générique (i,j) de la matrice de transition correspond à la probabilité de transiter vers l'état j à partir de l'état i. A quelles probabilités correspond la première ligne de la matrice de transition ? et celles de la dernière colonne ? Pour chaque lettre de l'alphabet, indiquer la transition la plus fréquente depuis cette lettre

La première ligne de la matrice de transition correspond aux probabilités de transiter vers l'état j à partir de l'état i=0. Cela correspond donc aux probabilités de transiter vers les différents états à partir de l'état de space qu'initie un mot.

La dernière colonne correspond aux probabilités de transiter vers l'état j=27 à partir de l'état i. Cela correspond donc aux probabilités de transiter vers l'état de space qui termine un mot quand on est dans un état i.

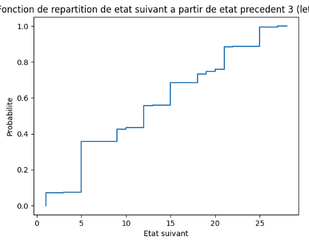
In [ ]:
```
for i in range(len(A)):
    current_letter = dic[str(i + 1)]
    index_max = np.argmax(A[i, :])
    max_value = A[i, index_max]
    max_letter = dic[str(index_max + 1)]
    print(f"From '{current_letter}' -> '{max_letter}' max probability {max_value}")
```

```
From ' ' -> 't' max probability 0.16452225
From 'a' -> 'n' max probability 0.22051689
From 'b' -> 'e' max probability 0.28275808
From 'c' -> 'o' max probability 0.16686338
From 'd' -> ' ' max probability 0.59884373
From 'e' -> ' ' max probability 0.36047379
From 'f' -> ' ' max probability 0.39653963
From 'g' -> ' ' max probability 0.31566736
From 'h' -> 'e' max probability 0.46971451
From 'i' -> 'n' max probability 0.24528243
From 'j' -> 'o' max probability 0.46979866
From 'k' -> ' ' max probability 0.37225637
From 'l' -> 'e' max probability 0.17086033
From 'm' -> 'e' max probability 0.26768727
From 'n' -> ' ' max probability 0.29421872
From 'o' -> 'n' max probability 0.16035077
From 'p' -> 'e' max probability 0.19473793
From 'q' -> 'u' max probability 0.96368715
From 'r' -> 'e' max probability 0.24477159
From 's' -> ' ' max probability 0.41030156
From 't' -> 'h' max probability 0.33937505
From 'u' -> 'r' max probability 0.15036937
From 'v' -> 'e' max probability 0.61843409
From 'w' -> 'a' max probability 0.20323878
From 'x' -> 't' max probability 0.20061728
From 'y' -> ' ' max probability 0.77582944
From 'z' -> 'e' max probability 0.55662188
From '.' -> '.' max probability 1.0
```

I.2.b

La fonction etat_suivant génère un état (à t+1) à partir de l'état courant (à t) et à l'aide de la matrice de transitions et de la fonction de répartition. Afficher sur un graphique la fonction de répartition pour une ligne de la matrice de transition et expliquer son rôle pour la génération de l'état à t+1.

In [ ]:
```
plt.step(range(1, 29), np.cumsum(A[2,]))
plt.title('Fonction de repartition de etat suivant a partir de etat precedent 3 (lettre b)')
plt.xlabel('Etat suivant')
plt.ylabel('Probabilite')
plt.show()
```



In [ ]:
```
def etat_suivant(ligne_matrice_trans):
    '''
    La fonction génère un état (à t+1) à partir de l'état courant (à t)
    La ligne de la matrice de transitions correspond aux probabilités
    de transiter de l'état courant vers les autres etats.
    '''
    f_repartition = np.cumsum(ligne_matrice_trans)
    unif = np.random.random()
    stat = 0
    while(unif >= f_repartition[stat]):
        stat = stat+1
    return stat+1
```

Ecrire la fonction genere_state_seq qui génère une séquence d'états jusqu'à aboutir à l'état final (28). Puis ecrire une fonction display_seq qui transforme une séquence d'états en séquence de caractères, à l'aide d'un dictionnaire. Utiliser ces fonctions pour générer des mots.

In [ ]:
```
def generate_state_seq(A):
    """
    Calculate a sequence of states

    """
    numStates = len(A)
    # Generate the emitting states sequence
    t = 0
    stateSeq = [1]

    while (stateSeq[-1] != numStates):
        curr_state_index = stateSeq[t] - 1
        curr_state_line = A[curr_state_index,]
        next_state = etat_suivant(curr_state_line)
        stateSeq.append(next_state)
        t += 1

    return stateSeq
```

In [ ]:
```
def display_seq(seq):
    str_seq = ""
    for i in seq:
        str_seq += dic[str(i)]
    return str_seq
```

In [ ]:
```
seq = generate_state_seq(A)
print(display_seq(seq))
```

```
ueders
```

I.2.c

On veut générer une suite de mots (phrase). Créer un état final de phrase (état 29, correspondant au caractère .) dont la probabilité de transition vers cet état depuis un état final de mot est 0.1. Ecrire une fonction modifie_mat_dic qui modifie la matrice de transition et le dictionnaire en conséquence. Donner des exemples de phrases générées.

In [ ]:
```
def modifie_mat_dic(A, dic):
    dic['29'] = '.'
    A_mod = np.zeros((29, 29))

    for i in range(28):
        for j in range(28):
            A_mod[i, j] = A[i, j]

    A_mod[27, 28] = 0.1
    A_mod[27, 27] = 0
    A_mod[27, 0] = 0.9

    A_mod[28, 28] = 1

    return A_mod, dic
```

In [ ]:
```
Amod, dicmod = modifie_mat_dic(A, dic)
```

In [ ]:
```
def genere_phrase(A, dic):
    t = 0
    numStates = len(dic)
    stateSeq = []
    # stateSeq est une liste
    stateSeq.append(1)  # on commence par etat 1

    while stateSeq[-1] != numStates:
        curr_state_index = stateSeq[t]-1
        curr_state_line = A[curr_state_index,]
        next_state = etat_suivant(curr_state_line)
        stateSeq.append(next_state)
        t += 1

    return stateSeq
```

In [ ]:
```
print('generation de phrases:')
for i in range (5) :
    state_seq=genere_phrase(Amod,dicmod)

    print(display_seq(state_seq))
```

```
generation de phrases:
f  mero  hotose  cor  n  lmime  ct  murmest  wat  hise  y  f  t  n  i  fide  jetowaust  or .
and  loy  ralvertoulmin  lmupeshulithasatrme  ty  w  rsey  g  s  as  ce  thond  gold  or  th  to  pr  fovef  putwatha  wiocat  neis  n  serela  me  tise  bowo  an .
g  aindmaseatsay  caretersspe  piseanchevioowbe  jupt  is  bomed  bye  dis  was  bl  be .
omangher  uans  ts .
e  rced  bunse .
```

I.3

In [ ]:
```
#transformer les espaces en +/- dans le dictionnaire et inversion
dicmod['1']='-'
dicmod['28']='+'
dicmod['29']='.'
```

Charger la matrice des transitions entre caractères pour le français. Ecrire une fonction calc_vraisemblance qui calcule la vraisemblance du modèle français pour une phrase donnée en multipliant les probabilités de transition. Pour tenir compte de toutes les transitions (notamment celles entre espaces de fin et de début de mots et vers la fin de phrase) on pourra transformer une séquence « mot1 mot2.» par la séquence « -mot1+-mot2-+.», les signes - , + et . représentant l'état initial de mot, l'état final de mot et l'état final de phrase, respectivement.

In [ ]:
```
def calc_vraisemblance(string, dic, mat):
    dicInv = {v: k for k, v in dic.items()}
    vraisemblance = 1.0
    cur_state = int(dicInv[string[0]])

    for i in range(1, len(string)):
        next_state = int(dicInv[string[i]])
        vraisemblance *= mat[cur_state - 1, next_state - 1]
        cur_state = next_state

    return vraisemblance
```

Calculer la vraisemblance des modèles français et anglais pour la phrase « to be or not to be ». De même calculer la vraisemblance des modèles français et anglais pour la phrase « etre ou ne pas etre ».

In [ ]:
```
sentence = "-to+-be+-or+-not+-to+-be+."

Fmod, dicmod = modifie_mat_dic(F, dic)
dicmod['1']='-'
dicmod['28']='+'
dicmod['29']='.'


print(f"Vraisemblance de la phrase '{sentence}':")
print(f"Anglais : {calc_vraisemblance(sentence, dicmod, Amod)}")
print(f"Francais : {calc_vraisemblance(sentence, dicmod, Fmod)}")



sentence2 = "-etre+-ou-+ne+-pas+-etre+."
print(f"Vraisemblance de la phrase '{sentence2}':")
print(f"Anglais : {calc_vraisemblance(sentence2, dicmod, Amod)}")
print(f"Francais : {calc_vraisemblance(sentence2, dicmod, Fmod)}")
```

```
Vraisemblance de la phrase '-to+-be+-or+-not+-to+-be+.':
Anglais : 8.1128922278094154e-28
Francais : 5.960208101868640e-30
Vraisemblance de la phrase '-etre+-ou-+ne+-pas+-etre+.':
Anglais : 4.462288711775253e-24
Francais : 1.145706887234709e-19
```