In [1]:

```python
import pandas as pd
import numpy as np
```

a) Warmup

In [2]:

```python
from collections import Counter
en_de = open("europarl-v7.de-en.lc.en", "r", encoding="utf-8").read()
en_fr = open("europarl-v7.fr-en.lc.en", "r", encoding="utf-8").read()
en_sv = open("europarl-v7.sv-en.lc.en", "r", encoding="utf-8").read()
en = en_de + " " + en_fr + " " + en_sv

de = open("europarl-v7.de-en.lc.de", "r", encoding="utf-8").read()
fr = open("europarl-v7.fr-en.lc.fr", "r", encoding="utf-8").read()
sv = open("europarl-v7.sv-en.lc.sv", "r", encoding="utf-8").read()

all_ = en + " " + de + " " + fr + " " + sv
```

In [3]:

```python
type(en)
```

Out[3]:

```
str
```

In [806]:

```python
Counter(en.split()).most_common(12)
```

Out[806]:

```
[('the', 58790),
 (',', 42043),
 ('.', 29542),
 ('of', 28406),
 ('to', 26842),
 ('and', 21459),
 ('in', 18485),
 ('is', 13331),
 ('that', 13219),
 ('a', 13090),
 ('we', 9936),
 ('this', 9916)]
```

In [807]:

```python
Counter(de.split()).most_common(12)
```

Out[807]:

```
[(',', 18549),
 ('die', 10521),
 ('.', 9733),
 ('der', 9374),
 ('und', 7028),
 ('in', 4175),
 ('zu', 3168),
 ('den', 2976),
 ('wir', 2863),
 ('daß', 2738),
 ('ich', 2670),
 ('das', 2669)]
```

In [808]:

```
Counter(fr.split()).most_common(13)
```

Out[808]:

```
[('&apos;', 16729),
 (',', 15402),
 ('de', 14520),
 ('la', 9746),
 ('.', 9734),
 ('et', 6619),
 ('l', 6536),
 ('le', 6174),
 ('les', 5585),
 ('à', 5500),
 ('des', 5232),
 ('que', 4797),
 ('d', 4555)]
```

In [809]:

```
Counter(sv.split()).most_common(13)
```

Out[809]:

```
[('.', 9648),
 ('att', 9181),
 (',', 8876),
 ('och', 7038),
 ('i', 5949),
 ('det', 5687),
 ('som', 5028),
 ('för', 4959),
 ('av', 4013),
 ('är', 3840),
 ('en', 3724),
 ('vi', 3211),
 ('jag', 3093)]
```

In [814]:

```
all_counter = Counter(all_.split())
```

The probabilities are found by taking the frequency of a given word and divide by the total number of words:

In [815]:

```
all_counter["speaker"]/sum(all_counter.values())
```

Out[815]:

```
1.9327394942430718e-05
```

In [816]:

```
all_counter["zebra"]/sum(all_counter.values())
```

Out[816]:

```
0.0
```

In [28]:

```
sentences = nltk.tokenize.sent_tokenize(all_)
```

In [29]:

```
sentences[0]
```

Out[29]:

```
'i declare resumed the session of the european parliament adjourned on friday 17 december 1999
, and i would like once again to wish you a happy new year in the hope that you enjoyed a pleas
ant festive period .'
```

b) Language modeling

If a word did not exist in the training data, its probability will be 0. This can be fixed by using laplace smoothing. This corresponds to initializing the frequency of all words to 1. A long sentence will lead to multiplying many small numbers together which will become very small. A solution is to us the log-probabilities instead.

In [3]:

```
import nltk
from collections import Counter, defaultdict
import math

def train_model(data):
    model = defaultdict(lambda: defaultdict(lambda: 0))

    # Split data into sentences
    sentences = data.splitlines()

    # For each sentence, compute frequency of w2 arriving after w1
    for sentence in sentences:
        for w1, w2 in nltk.bigrams(sentence.split()[:-1], pad_right=True, pad_left=True):
            model[w1][w2] += 1
    return model
```

In [4]:

```
model = train_model(en_sv)
def language_model(sentence, model):
    words = sentence.split()
    start_count = float(sum(model[None].values()))
    prob = math.log( (model[None][words[0]] + 0.001) / (start_count+len(model[None])*0.001)) # laplace lambda smoothing on first word
    #prob = math.log( (model[None][words[0]]) / (start_count)) # laplace lambda smoothing on first word
    #prob = (model[None][words[0]]) / (start_count)
    for w1, w2 in nltk.bigrams(words, pad_right=False, pad_left=False):
        #print(w1,w2)
        #print(total_count)
        prob_now = 0
        total_count = float(sum(model[w1].values()))
        p = (model[w1][w2]+0.001) / (total_count+len(model[w1])*0.001) # laplace lambda smoothing
        #p = (model[w1][w2]) / (total_count) # laplace lambda smoothing
        #model[w1][w2] = (model[w1][w2]) / (total_count) # laplace lambda smoothing
        prob = prob + math.log(p) # Log probabilities
        #prob = prob * model[w1][w2]
    return prob
```

In [6]:

```
math.exp(language_model("our member states", model))
```

Out[6]:

```
2.002361613074967e-05
```

In [7]:

```
math.exp(language_model("member our states", model))
```

Out[7]:

```
1.4234695149217394e-12
```

We can now compute $P(E)$

c) Translation modeling

In [35]:

```
#def run_em(lang1, lang2, n):
import collections
lang1 = sv
lang2 = en_sv
#lang1 = en_sv
#lang2 = sv
n = 100

n_words = len(set(lang2.split()))
lang1 = lang1.splitlines()
lang2 = lang2.splitlines()
#t = collections.defaultdict(lambda: 1/n_words)
t = defaultdict(lambda: defaultdict(lambda: 1/n_words))

# E/M algorithm
for i in range(n):
    corpus = zip(lang1, lang2)
    count = collections.defaultdict(float)
    total = collections.defaultdict(float)
    s_total = collections.defaultdict(float)
    for (l1, l2) in corpus:
        # compute normalization

        l1 = l1.split()
        l2 = l2.split()
        # Insert null word at start
        l2.insert(0,"NULL")
        #print(l1)

        for f in l1:
            s_total[f] = 0.0
            for e in l2:
                s_total[f] += t[f][e]

        for f in l1:
            for e in l2:
                delta = t[f][e] / s_total[f]
                count[(e, f)] += delta
                total[e] += delta

    # estimate probability
    for (f, e) in count.keys():
        #if count[(f, e)] == 0:
            #print(f, e, count[(f, e)])
        t[f][e] = count[(f, e)] / total[e]

    #print(Counter(t["europeiska"]).most_common(10))
    # Find the words that are most likely to align with the word european
    test = []
    for w in t:
        #prob = ws["european"]
        prob = t[w]["european"]
        test.append((w, prob))
    print(sorted(test, key=lambda x: x[1], reverse=True)[0:10])
    #print("\n")
    #print("test")
    #return t
```

```
[('.', 0.03591034653913155), (',', 0.03372474830401548), ('att', 0.03332533292239441), ('och',
0.028216364762232687), ('i', 0.027025686545253), ('europeiska', 0.02597178687247437), ('det', 0
.020548575860822177), ('för', 0.01956648122988557), ('som', 0.018686658041077588), ('en', 0.017
656463635023014)]
[('europeiska', 0.16755995477109858), ('.', 0.05111446624924288), (',', 0.047113972999060276),
('att', 0.044936728607888374), ('i', 0.0439457468882919), ('och', 0.040990894746709354), ('en',
0.028060265892112347), ('för', 0.027456514082219212), ('unionen', 0.02710952557008033), ('det',
0.027010613200927517)]
[('europeiska', 0.3999353271255518), ('i', 0.04029466146199781), ('.', 0.03957039283243889), ('
,', 0.03726033558457288), ('att', 0.03416472425393751), ('och', 0.033111329979174), ('europeisk
', 0.03222043498307792), ('unionen', 0.029840331102603732), ('en', 0.025488676146094496), ('den
', 0.024758186528307996)]
[('europeiska', 0.5716593290434614), ('europeisk', 0.042219696757371), ('i', 0.0330475088863122
7), ('.', 0.027757692310377877), (',', 0.026510108776129665), ('att', 0.023739155629942), ('och
', 0.023457943339268844), ('den', 0.02177245693804311), ('unionen', 0.02137438286026325), ('eur
opaparlamentet', 0.02048948464529695)]
[('europeiska', 0.676669594402242), ('europeisk', 0.04964457030230529), ('i', 0.027276598023511
307), ('.', 0.01968215059283706), ('den', 0.019173679875087803), (',', 0.019068200355659613), (
'att', 0.01685533659286558), ('och', 0.01663107704012793), ('europaparlamentet', 0.01612231160
5604907), ('en', 0.01582854903548991)]
```

[('europeiska', 0.7413889908540344), ('europeisk', 0.05588425477831991), ('i', 0.02312092461004
659), ('den', 0.01720449170180861), ('.', 0.01432553713285935), (',', 0.01414410455249796), ('e
n', 0.012861950263567817), ('att', 0.012361279791623724), ('och', 0.012162459390752042), ('euro
paparlamentet', 0.01186078435014985)]
[('europeiska', 0.7828857748022445), ('europeisk', 0.06136366536565766), ('i', 0.02008041912146
8834), ('den', 0.01566294351565134), ('en', 0.010803311591158642), (',', 0.010792955419502375),
('.', 0.01066212107505383), ('att', 0.009303068302841354), ('och', 0.009194441436997181), ('eur
opaparlamentet', 0.008458733359257023)]
[('europeiska', 0.8103736018361043), ('europeisk', 0.0662075710855376), ('i', 0.017772761389869
24), ('den', 0.01438910898624399), ('en', 0.00932451506569945), (',', 0.008426701480569584), ('
.', 0.008071304852094367), ('och', 0.007157031187106833), ('att', 0.0071366101175515265), ('eur
opaparlamentet', 0.005942847824196722)]
[('europeiska', 0.829110538740317), ('europeisk', 0.07047845467735513), ('i', 0.015956265110397
515), ('den', 0.01329508124398976), ('en', 0.008215295800391813), (',', 0.006700456693823119),
('.', 0.006190319917968365), ('och', 0.00570986370697236), ('att', 0.005553954819842967), ('til
l', 0.005205722411667961)]
[('europeiska', 0.8422609571174579), ('europeisk', 0.07422650340065175), ('i', 0.01447743708681
0312), ('den', 0.01233393763519077), ('en', 0.007346652372925064), (',', 0.005405578426447147),
('.', 0.004797030910018389), ('till', 0.004686005898477835), ('och', 0.004648178740809186), ('a
tt', 0.004370668143854596)]
[('europeiska', 0.8517605876615674), ('europeisk', 0.07750525544534254), ('i', 0.01323641351294
2259), ('den', 0.011478619902817419), ('en', 0.006640142708036643), (',', 0.004411213718592315)
, ('till', 0.004261508038680692), ('och', 0.003845682193471899), ('.', 0.0037485681154658407),
('att', 0.0034699197472872756)]
[('europeiska', 0.858814061265775), ('europeisk', 0.08037127683439747), ('i', 0.012167296240388
705), ('den', 0.010711480779359649), ('en', 0.00604782812258614), ('till', 0.00390552467805167)
, (',', 0.0036327529897047534), ('och', 0.0032225127705783957), ('.', 0.0029494596299357214), (
'att', 0.002774338032868995)]
[('europeiska', 0.8641846205958541), ('europeisk', 0.08288155242956237), ('i', 0.01122629206020
729), ('den', 0.01001928300305469), ('en', 0.00553968688153814), ('till', 0.0036003222631670767
), (',', 0.003013678229965546), ('och', 0.0027269875176008786), ('.', 0.0023339851237588567), (
'för', 0.0022884354177243825)]
[('europeiska', 0.8683644484776324), ('europeisk', 0.08508984583035253), ('i', 0.01038423099629
007), ('den', 0.009391191750099354), ('en', 0.005096253511353947), ('till', 0.00333389043886274
3), (',', 0.0025149677234785972), ('och', 0.0023249742269168144), ('för', 0.001932194142339005)
, ('.', 0.0018558215067275056)]
[('europeiska', 0.8716841552266698), ('europeisk', 0.08703743036859997), ('i', 0.00962166759575
6547), ('den', 0.008818460101401308), ('en', 0.004704617894469802), ('till', 0.0030979587892866
51), (',', 0.0021089351998627892), ('och', 0.0019935029482617377), ('för', 0.001635525319613959
5), ('.', 0.0014816675576300386)]
[('europeiska', 0.8743697139400727), ('europeisk', 0.08875776903891515), ('i', 0.00892539570672
2674), ('den', 0.008294484087585161), ('en', 0.004356094126102736), ('till', 0.0028866805058659
645), (',', 0.001775466782859341), ('och', 0.001716741392899413), ('för', 0.0013872079619136968
), ('att', 0.0011888356789146455)]
[('europeiska', 0.8765753336315741), ('europeisk', 0.09028200974363815), ('i', 0.00828607401128
1334), ('den', 0.007814381677559067), ('en', 0.004044545971558291), ('till', 0.0026957780676292
206), (',', 0.0014996243893461159), ('och', 0.0014834261187828535), ('för', 0.00117854981341182
63), ('om', 0.001032535067484841)]
[('europeiska', 0.8784098255968109), ('europeisk', 0.09163752476552176), ('i', 0.00769681480362
8078), ('den', 0.007374083708140711), ('en', 0.003765180623716327), ('till', 0.0025220228557581
753), ('och', 0.0012852663217548578), (',', 0.0012700926289866328), ('för', 0.00100267494178597
98), ('om', 0.0009167849661938376)]
[('europeiska', 0.8799528359411611), ('europeisk', 0.09284754112643466), ('i', 0.00715242161540
1665), ('den', 0.006969553623365269), ('en', 0.003513913623399362), ('till', 0.0023629069274927
043), ('och', 0.0011159726799292139), (',', 0.0010781483554178556), ('för', 0.00085405549750534
84), ('om', 0.0008157315397338126)]
[('europeiska', 0.8812641752210326), ('europeisk', 0.09393158661684788), ('i', 0.00664894796040
0878), ('den', 0.006596499304729936), ('en', 0.0032871645183533485), ('till', 0.002216439563085
6824), ('och', 0.0009706590858290876), (',', 0.0009169651167587169), ('för', 0.0007281971068074
637), ('om', 0.0007271466351933387)]
[('europeiska', 0.8823896991761078), ('europeisk', 0.09490587723088738), ('den', 0.006250416855
361504), ('i', 0.00618329049237557), ('en', 0.0030818211797635645), ('till', 0.0020810207635308
287), ('och', 0.000845457336834619), (',', 0.0007811338409227075), ('om', 0.0006491987208383731
), ('för', 0.0006214178214160781)]
[('europeiska', 0.8833653190226205), ('europeisk', 0.0957837712824936), ('den', 0.0059267818127
76672), ('i', 0.005752782463969414), ('en', 0.0028952168850203104), ('till', 0.0019553552957310
78), ('och', 0.0007372567118964675), (',', 0.0006663243389499405), ('om', 0.000580376012097849)
, ('för', 0.0005306836224137623)]
[('europeiska', 0.8842195941103675), ('europeisk', 0.09657644133814808), ('den', 0.005621369532
842496), ('i', 0.005354919454426184), ('en', 0.002725079898592508), ('till', 0.0018383868929936
814), ('och', 0.0006435214283600661), (',', 0.0005690395000285566), ('om', 0.000519425275901546
2), ('för', 0.00045348159253640074)]
[('europeiska', 0.884975131916147), ('europeisk', 0.09729353406954029), ('den', 0.0053307051887
13848), ('i', 0.004987258584569747), ('en', 0.002569465770812944), ('till', 0.00172924519234538
71), ('och', 0.0005621599119672977), (',', 0.0004864316938267013), ('om', 0.0004653019518117702
4), ('för', 0.00038772055797508837)]
[('europeiska', 0.8856493851589224), ('europeisk', 0.09794355639073148), ('den', 0.005052486569
007941), ('i', 0.004647424754166952), ('en', 0.0024266889224671797), ('till', 0.001627204004321
9596), ('och', 0.0004914301743241965), ('om', 0.00041712976557568154), (',', 0.0004161638204084
728), ('för', 0.00033165252283590515)]

```
[('europeiska', 0.8862554683688497), ('europeisk', 0.09853401741100326), ('den', 0.004785697747
553504), ('i', 0.004333150910956852), ('en', 0.0022952672602444832), ('till', 0.001531649472028
631), ('och', 0.0004298696720071428), ('om', 0.00037416855224456087), (',', 0.00035630424786726
82), ('för', 0.0002838101602965)]
[('europeiska', 0.8868031876974445), ('europeisk', 0.09907149631729208), ('den', 0.004530292393
825691), ('i', 0.00404231327869756), ('en', 0.0021738907880490825), ('till', 0.0014420558919249
575), ('och', 0.0003762407285802964), ('om', 0.00033578881075579177), (',', 0.00030524722110179
62), ('för', 0.00024295735690068319)]
[('europeiska', 0.8873000836753715), ('europeisk', 0.09956173026429906), ('den', 0.004286654130
180804), ('i', 0.0037729504762375916), ('en', 0.0020614174094274397), ('till', 0.00135796739915
57305), ('och', 0.0003294870442073144), ('om', 0.00030145160806513795), (',', 0.000261651686643
733), ('för', 0.0002080504738409558)]

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-35-528cafb77057> in <module>
     40 
     41     # estimate probability
---> 42     for (f, e) in count.keys():
     43         #if count[(f, e)] == 0:
     44             #print(f, e, count[(f, e)])

KeyboardInterrupt: 
```

In [42]:

```
def translation_model(in_sentence, out_sentence, t):
    words = in_sentence.split()
    words2 = out_sentence.split()
    prob = 0
    for w1 in words:
        max_prob = 0
        for w2 in words2:
            temp_prob = t[w1][w2]
            if temp_prob > max_prob:
                max_prob = temp_prob
        #print(max_prob)
        prob += math.log(max_prob) #+ math.log(1/len(out_sentence))
    return prob
```

In [43]:

```
translation_model("jag är bra", "i is good", t)
```

Out[43]:

```
-1.0253785500545938
```

In [11]:

```
translation_model("jag är bra", "i is bad", t)
```

Out[11]:

```
-12.550998882407782
```

The above function computes $P(F|E, A)$

d) Decoding

The most simple case: Only consider translation model based on most probable alignment

In [12]:

```
def simple_decode(sentence, t):
    words = sentence.split()
    eng = ""
    for word in words:
        if len(t[word]) == 0:
            eng += word + " "
        else:
            eng += Counter(t[word]).most_common(1)[0][0] + " "
    return eng.strip()
```

In [18]:

```
Counter(t["i"]).most_common(3)
```

Out[18]:

```
[('in', 0.7017256406325446),
 ('into', 0.4937823209075722),
 ('treaty', 0.397950042591096 84)]
```

In [19]:

```
simple_decode("mycket bra är jag", t)
```

Out[19]:

```
'very good is i'
```

In [20]:

```
simple_decode("jag är mycket bra", t)
```

Out[20]:

```
'i is very good'
```

In [24]:

```
simple_decode("herr talman jag", t)
```

Out[24]:

```
'mr president i'
```

In [25]:

```
simple_decode("våra medlemsstater", t)
```

Out[25]:

```
'our states'
```

The implementation below will construct a set of candidate translations based on the 3 msot probably alignments for each word in the swedish sentence. It will pick the sentence that maximizes $P(E) * P(F \mid E)$.

In [39]:

```
from itertools import permutations

def permute(l):
    if len(l) == 1:
        return l[0]
    else:
        lnew = []
        for a in l[0]:
            for b in permute(l[1:]):
                lnew.append(a+" " + b)
        return lnew

def decode(swe, t, model):
    words = swe.split()
    candidate_sentences = []
    # Find the 3 english words that are most probable to be aligned with each swedish word
    for word in words:
        sentence = []
        mc = Counter(t[word]).most_common(3)
        for i,_ in mc:
            sentence.append(i)
        candidate_sentences.append(sentence)
    perm_sentences = permute(candidate_sentences)

    max_prob = -np.inf
    best_sentence = "UNKNOWN"
    for s in perm_sentences:
        perm = list(permutations(s.split()))
        for p in perm:
            eng_sentence = ' '.join(p)
            prob = language_model(eng_sentence, model) + translation_model(swe, eng_sentence, t)
            if prob >= max_prob:
                max_prob = prob
                best_sentence = eng_sentence
    return best_sentence
```

In [39]:

```
Counter(t["domstolen"]).most_common(5)
```

Out[39]:

```
[('court', 0.2908739985965263),
 ('acquitted', 0.15787944446357263),
 ('referrals', 0.026387736694697846),
 ('non-application', 0.026387736694697846),
 ('non-transposition', 0.026387736694697846)]
```

In [28]:

```
decode("jag är bra", t, model)
```

Out[28]:

```
'i am good'
```

In [29]:

```
decode("jag är mycket bra", t, model)
```

Out[29]:

```
'i am very good'
```

In [30]:

```
decode("mycket bra är jag", t, model)
```

Out[30]:

```
'i am very good'
```

In [31]:

```
decode("jag mycket är bra", t, model)
```

Out[31]:

```
'i am very good'
```

In [32]:

```
decode("våra medlemsstater", t, model)
```

Out[32]:

```
'our member'
```

In [33]:

```
decode("jag vill vet om jag", t, model)
```

Out[33]:

```
'i know if i want'
```

In [35]:

```
decode("domstolen har friat honom", t, model)
```

Out[35]:

```
'have acquitted acquitted him'
```

In [36]:

```
simple_decode("domstolen har friat honom", t)
```

Out[36]:

```
'court has acquitted him'
```

In [1010]:

```
language_model("our states", model)
```

Out[1010]:

```
-11.130075819478984
```

In [1085]:

```
translation_model("våra medlemsstater", "our member", t)
```

Out[1085]:

```
-2.8656150459222722
```

In [1086]:

```
translation_model("våra medlemsstater", "our states", t)
```

Out[1086]:

```
-2.600741149878915
```

In [ ]:

```

```