ASSIGNMENT-5

SOLUTIONS BY HARSHITA JHAVAR
Matriculation Number ? 2566267

Part 2.)

The Alignment Error Rate of the baseline system is 0.681684 with Precision = 0.243110 and Recall = 0.544379 .

Part 3.)
Source Code: Please see /IBM_Model1/IBM_Model1.py
Outputs: Please see /IBM_Model1/output_IBM_Model1

The results after running the IBM Model 1 are:
Precision = 0.212239
Recall = 0.674556
Alignment Error Rate = 0.734247

Part 4.) GIZA++
The original output files for each of the models generated after 10 iterations can be found in th e respective folders of Model 1 (’Giza-model1-outputs’) and Model 2 of GIZA (’Giza-model2-outputs ’).

I used this command to build giza for model 1 after 10 iterations
./GIZA++ -S ./english.vcb -T ./french.vcb -C ./english_french.snt -model1iterations 10 -model2ite rations 0 -model3iterations 0 -model4iterations 0 -model5iterations 0 -hmmiterations 0 -t1 10 -co mpactalignmentformat 1 -outputpath ./output

I used this command to build giza for model 2 after 10 iterations
./GIZA++ -S ./english.vcb -T ./french.vcb -C ./english_french.snt -model1iterations 0 -model2iter ations 1 -model3iterations 0 -model4iterations 0 -model5iterations 0 -hmmiterations 0 -t1 10 -com pactalignmentformat 1 -outputpath ./output

Evaluation of GIZA++:
Note: For evaluation, I changed the reports.cpp file line 127 and included ’-’ in it, and set "mo del1dumpfrequency" and "compactalignmentformat", both equal to 1 during the call at Terminal. I a lso inverted the indexes and tried for getting an output in the form which is acceptable for scor ing alignment as input but I was not able to get the required output.
I have submitted the final output I got for model 1 and model 2 but could not compute its accurac y from the score-alignment python file.

Alignment example:

Sentence ID - 31
Original Sentence
french ils veulent de les résultats .
english they want to see results .
Baseline System= 0-0 1-1 2?4 3?4 4-4 5-5
IBM Model 1 ==== 0-0 0-0 0-0 0-0 0-0 0-0 0-1 1-1 1-1 1-1 1-1 1-1 0-2 1-2 1-2 1-2 1-2 1-2 0-3 1-3 1-3 1-3 4-3 4-3 0-4 1-4 1-4 1-4 4-4 4-4 0-5 1-5 1-5 1-5 4-5 5-5
GIZA ++ Model1== 0 0 1 1 2 2 4 3 4 4 5 5

```
import optparse
import sys
from collections import defaultdict


def trainfw(itr, q):
        x=0


        # Initialize probablility set of last iteration to 1
        #    * for tracking convergence
        for (f, e) in bitext:
                        for f_i in f:
                                for e_j in e:
                                        oldtef.append(1.0)



        while 0.00001< compT_EF(oldtef, t_ef, x) and x!=itr:
                # zero totals and counts for this iteration
                for f in f_total:
                        f_total[f]=0.0
                for (e,f) in ef_count:
                        ef_count[(e,f)]=0.0

                for (f,e) in bitext:
                        for e_j in e:
                                se_total=0.0
                                for f_i in f:
                                        se_total += t_ef[(e_j,f_i)]  * len(e)
                                for f_i in f:
                                        ef_count[(e_j,f_i)] += (t_ef[(e_j,f_i)] * len(e) * len(f)
)/se_total
                                        f_total[f_i] += (t_ef[(e_j,f_i)] * len(e) * len(f))/se_to
tal

                print ("before dict", x )


                # calculate new probablilities
                for e_j,f_i in t_ef:
                        t_ef[(e_j,f_i)] = ef_count[(e_j,f_i)]/f_total[f_i]

                x+=1
                # write output for the current iteration
                writeOutput('interfw-'+str(q)+"-"+str(x))

        # write output for the current set of iterations
        writeOutput('finalfw-'+str(q)+"-"+str(x))


def compT_EF(t1, t2, x):
        # compute convergence ie prev set of probs(oldtef) vs current tef
        if x == 0:
                return 1
        global oldtef
        global t_ef
        tot=0.0
        cnt=0
        c2=0
        print ("len tef",len(t_ef))
        print ("len oldtef",len(oldtef))
        for s in t_ef:
                if t_ef[s] > 0:
                        if c2<5:
                                print( s, c2, oldtef[cnt], t_ef[s], oldtef[cnt]-t_ef[s])
                        tot+=abs(oldtef[cnt]-t_ef[s])
                        c2+=1
                cnt+=1

        print ("conv?:", tot/c2)
```

```
        oldtef=[]
        for e in t_ef:
                oldtef.append(t_ef[e])
        return (tot/c2)

def writeOutput(x):
        global bitext
        global t_ef
        outfile = open(opts.out+'.1.'+x+'.a','w')
        #printing to file
        for (f, e) in bitext:
          for (j, e_j) in enumerate(e):
                  best_p=0
                  best_i=0
                  for (i, f_i) in enumerate(f):
                        if t_ef[(e_j,f_i)] > best_p:
                                best_p=t_ef[(e_j,f_i)]
                                best_i=i
                        outfile.write("%i-%i " % (best_i,j))
          outfile.write("\n")
        outfile.close()
        return

if __name__ == "__main__":
        optparser = optparse.OptionParser()

        optparser.add_option("-d", "--data", dest="train", default="data/hansards", help="Data fi
lename prefix (default=data)")
        optparser.add_option("-e", "--english", dest="english", default="e", help="Suffix of Engl
ish filename (default=e)")
        optparser.add_option("-o", "--out", dest="out", default="m1.", help="output prefid (defau
lt=m1.)")
        optparser.add_option("-f", "--french", dest="french", default="f", help="Suffix of French
 filename (default=f)")
        optparser.add_option("-t", "--threshold", dest="threshold", default=0.5, type="float", he
lp="Threshold for aligning with Dice's coefficient (default=0.5)")
        optparser.add_option("-n", "--num_sentences", dest="num_sents", default=sys.maxsize, type
="int", help="Number of sentences to use for training and alignment")
        (opts, _) = optparser.parse_args()

        f_data = "%s.%s" % (opts.train, opts.french)
        e_data = "%s.%s" % (opts.train, opts.english)
        bitext = [[sentence.strip().split() for sentence in pair] for pair in list(zip(open(f_dat
a), open(e_data)))[:opts.num_sents]]
        f_total = defaultdict(float)
        ef_count = defaultdict(float)

        #reversing
        e_total = defaultdict(float)
        fe_count = defaultdict(float)

        #global probs
        t_ef = defaultdict(float)

        oldtef=[]

        # uniformly initialize all probablilities
        for (n, (f, e)) in enumerate(bitext):
                for e_j in e:
                        for f_i in f:
                                t_ef[(e_j,f_i)]=1.0


        trainfw(1,1)
```