Traitement automatique du langage
TP 3 — Identify gene names with HMMs

Yves Scherrer

Convert training data by replacing rare words

```
#! /usr/bin/env python3

import sys, collections

wordfreqs = collections.defaultdict(int)
f = open(sys.argv[1], 'r')
for line in f:
    line = line.strip()
    if line != "":
        wordfreqs[line.split(" ")[0]] += 1
rarewords = [w for w in wordfreqs if wordfreqs[w] < 5]


f.seek(0)   # go back to the beginning of the file
of = sys.stdout
for line in f:
    line = line.strip()
    if line == "":
        of.write("\n")
    else:
        elements = line.split(" ")
        if elements[0] in rarewords:
            if len(elements) > 1:
                of.write("{} {}\n".format("_RARE_", elements[1]))
            else:
                of.write("_RARE_\n")
        else:
            of.write(line + "\n")
f.close()
of.close()
```

Unigram tagger

```
#! /usr/bin/env python3

import sys

words = set()
tags = set()
e = {}

def collectCounts(infilename):
    wordtagcounts = {}
    tagcounts = {}

    f = open(infilename, 'r')
    for line in f:
        elements = line.strip().split(' ')
        if elements[1] == "WORDTAG":
            wordtagcounts[(elements[3], elements[2])] = int(elements
               [0])
            words.add(elements[3])
        elif elements[1] == "1-GRAM":
            tagcounts[elements[2]] = int(elements[0])
            tags.add(elements[2])
    f.close()

    for (word, tag) in wordtagcounts:
        e[(word, tag)] = wordtagcounts[(word, tag)] / tagcounts[tag]


def sentenceIterator(filehandle):
    currentSentence = [] #Buffer for the current sentence
    for l in filehandle:
        l = l.strip()
        if l == "":
            if currentSentence:
                yield currentSentence
                currentSentence = []
            else:
                sys.stderr.write("WARNING: Got empty input file/
                   stream.\n")
                raise StopIteration
        else:
            currentSentence.append(l)
    if currentSentence:
        yield currentSentence
```

```
def tagFile(infilename, outfilename):
    infile = open(infilename, 'r')
    outfile = open(outfilename, 'w')
    for sentence in sentenceIterator(infile):
        for word in sentence:
            maxProb = 0.0
            maxClass = ""
            for c in tags:
                if word not in words:
                    emissionProb = e[("_RARE_", c)]
                elif (word, c) not in e:
                    emissionProb = 0
                else:
                    emissionProb = e[(word, c)]
                if emissionProb > maxProb:
                    maxProb = emissionProb
                    maxClass = c
            outfile.write("{} {}\n".format(word, maxClass))
        outfile.write("\n")
    infile.close()
    outfile.close()

if __name__ == "__main__":
    collectCounts(sys.argv[1])
    tagFile(sys.argv[2], sys.argv[3])
```

Results

```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

        precision       recall          F1-Score
GENE:   0.158861        0.660436        0.256116
```

Trigram HMM tagger

```
#! /usr/bin/env python3

import sys

tags = set()
words = set()
q = {}
e = {}


def collectCounts(infilename):
    wordtagcounts = {}
    unigramcounts = {}
    bigramcounts = {}
    trigramcounts = {}

    f = open(infilename, 'r')
    for line in f:
        elements = line.strip().split(' ')
        if elements[1] == "WORDTAG":
            wordtagcounts[(elements[3], elements[2])] = int(elements
               [0])
            words.add(elements[3])
        elif elements[1] == "1-GRAM":
            unigramcounts[elements[2]] = int(elements[0])
        elif elements[1] == "2-GRAM":
            bigramcounts[(elements[2], elements[3])] = int(elements
               [0])
        elif elements[1] == "3-GRAM":
            trigramcounts[(elements[2], elements[3], elements[4])] =
                int(elements[0])
            tags.add(elements[2])
            tags.add(elements[3])
            tags.add(elements[4])
    f.close()

    for (word, tag) in wordtagcounts:
        e[(word, tag)] = wordtagcounts[(word, tag)] / unigramcounts[
           tag]
    for tag in tags:
        for tag_1 in tags:
            for tag_2 in tags:
                if (tag_2, tag_1, tag) in trigramcounts:
                    q[(tag, tag_2, tag_1)] = trigramcounts[(tag_2,
```

```
                        tag_1, tag)] / bigramcounts[(tag_2, tag_1)]


def sentenceIterator(filehandle):
    currentSentence = [] #Buffer for the current sentence
    for l in filehandle:
        l = l.strip()
        if l == "":
            if currentSentence:
                yield currentSentence
                currentSentence = []
            else:
                sys.stderr.write("WARNING: Got empty input file/
                   stream.\n")
                raise StopIteration
        else:
            currentSentence.append(l)
    if currentSentence:
        yield currentSentence


def viterbi(sentence):
    # initializations
    n = len(sentence)
    pi = {}
    pi[(-1, "*", "*")] = 1
    bp = {}
    S = {}
    S[-2] = ["*"]
    S[-1] = ["*"]
    for k in range(n):
        S[k] = tags

    # filling the viterbi probability table and backpointer table
    for k in range(n):
        if sentence[k] in words:
            word = sentence[k]
        else:
            word = "_RARE_"

        for u in S[k-1]:
            for v in S[k]:
                maxProb = 0
                maxClass = ""
                for w in S[k-2]:
                    prob = pi.get((k-1, w, u), 0) * q.get((v, w, u),
                        0) * e.get((word, v), 0)
```

```
                    if prob > maxProb:
                        maxProb = prob
                        maxClass = w
                if maxProb > 0:
                    pi[(k, u, v)] = maxProb
                    bp[(k, u, v)] = maxClass

    # final probabilities
    maxProb = 0
    maxU, maxV = "", ""
    for u in S[n-2]:
        for v in S[n-1]:
            prob = pi.get((n-1, u, v), 0) * q.get(("STOP", u, v), 0)
            if prob > maxProb:
                maxProb = prob
                maxU, maxV = u, v

    # creating the tag table
    y = {}
    if maxU == "":
        y[n-2] = "O"
    else:
        y[n-2] = maxU
    if maxV == "":
        y[n-1] = "O"
    else:
        y[n-1] = maxV
    for k in range(n-3, -1, -1):
        try:
            y[k] = bp[(k+2, y[k+1], y[k+2])]
        except KeyError:
            print("Underflow in sentence: {}".format(" ".join(
               sentence)))
            print("Returning 'O' instead")
            y[k] = 'O'
    return [y[k] for k in sorted(y)]


def tagFile(infilename, outfilename):
    infile = open(infilename, 'r')
    outfile = open(outfilename, 'w')
    for sentence in sentenceIterator(infile):
        tagSequence = viterbi(sentence)
        for i in range(len(sentence)):
            outfile.write("{} {}\n".format(sentence[i], tagSequence[
               i]))
        outfile.write("\n")
```

```
    infile.close()
    outfile.close()


if __name__ == ”__main__”:
    collectCounts(sys.argv[1])
    tagFile(sys.argv[2], sys.argv[3])
```

Results

```
Found 373 GENEs. Expected 642 GENEs; Correct: 202.

        precision       recall          F1-Score
GENE:   0.541555        0.314642        0.398030
```