Advanced NLP - Assignment No 2

- Rushabh Dharia, Chaitanya Patil, Animesh Sagar, Pranay Shah

(a) We've selected the following novels
  (i) A House of Pomegranates by Oscar Wilde
  (ii) A Christmas Carol by Charles Dickens

(b) Split the above novels in chunks of 1000 lines using the split command in linux split -l 1000 filename. Generated 4 files for each novel and generated a corresponding XML file for each of those files. Stored all of them in folder b.

(c) Speech verb actor pattern is searched

The key verbs which identify the conversations are say, speak, talk, ask, reply, answer, add, continue, go on, cry, sigh, and think. If a verb from the above short list cannot be found, any verb that is preceded by a name or a personal pronoun in the vicinity of the utterance is selected as the speech verb.
Dependency relations in the text are identified using the machine based parser, such as dobj and advmod. The method (Referred from the papers referenced) extracts the speaker's name from the dependency relation nsubj, which links a speech verb with a noun phrase that is the syntactic subject of a clause. Once an explicit speaker's name or an anaphoric expression is located, we determine the corresponding gender information by referring to the character list or by following straightforward rules to handle the anaphora

d) Our CoreNLP processing pipeline was run over "A Christmas Carol by Charles Dickens" for testing purposes. The pipeline correctly identified the frequencies with which the characters talk to each other. The most common interactions involved a character called Ebenezer Scrooge. Below is a table which shows the top 5 interactions from the novel.

Note: Character 1 and Character 2 are in no specific order, i.e we are counting interactions between characters, and not who told a dialogue to whom.

Character 1	**Character 2**	**Frequency**
Ebenezer Scrooge	Ghost of Christmas Yet to Come	34
Ebenezer Scrooge	Ghost of Christmas Past	15
Ebenezer Scrooge	Ghost of Christmas Present	77
Ebenezer Scrooge	Bob Cratchit	26
Ebenezer Scrooge	Tiny Tim	26

e) Our model has counted interactions, for example, between "miser and some character 2". Each of these have some frequency. But the model has to figure out that the miser or old miser as in the novel, is actually Ebenezer Scrooge. For this, we need to have something

which stores the descriptions for a character, or something that remembers how a character can be referred to. Basically, we need a model that can remember past data. To us, the first thing that comes to mind, is a neural network like LSTM, or a GRU.

f) Submitted as a separate file.

(g)

Matrix multiplier

Result of the evaluation $C = AB^4$

$$A = \begin{pmatrix} 0.5 & 0.3 & 0.2 & 0.0 \end{pmatrix} \quad B = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.1 \\ 0.3 & 0.4 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.3 & 0.5 \end{pmatrix} \quad C = \begin{pmatrix} 0.18791 & 0.33393 & 0.27332 & 0.20484 \end{pmatrix}$$

(g 1) What is the probability that after 4 steps exactly 3 lines are busy?
A1) - 0.20484

(g 2) What number of lines being busy has the highest probability after 4 steps?
A2) 1 line

h)

Viterbi Decoding-Tracking Algorithm

AT-NC				0	
NIL				4.90E-17	
AT-HL				2.70E-16	
CC				1.04E-12	
AT				1.90E-09	
VB-HL			7.60E-13		
JJ			1.15E-08		
IN			6.40E-09		
CS			2.0063-9		
VBZ-HL		3.73E-10			
VBL		9.96E-07			
NNs		2.62E-06			
VB	4.30E-07		6.85E-09		
NN-HL	3.59E-07				
NN	0.00117315				2.03E-11
Words	**Time**	**Flies**	**Like**	**an**	**arrow**

P(Time|NN) = 0.99		P(Files|NNS) = 0.636		P(like|CS) = 0.77		P(an|AT) = 0.99		P(arrow|NN) = 1
P(Time|NN-HL) = 0.005		P(Files|VBZ) = 0.27		P(like|VB) = 0.169		P(an|CC) = 0.0031		
P(Time|VR) = 0.0006		P(Files|VBZ-HL) = 0.09		P(like|JJ) = 0.02		P(an|AT-HL) = 0.0028		
				P(like|JN) = 0.025		P(an|NIC) = 0.0005		
				P(like|VB-HL) = 0.008		P(an|AT-NC) = 0.0002		

Time: NN
Flies: NNS
LIke: JJ
An: AT
Arrow : NN

I)**A Maximum Entropy Model for Part-Of-Speech Tagging**

Corpus: >>> Part-of-speech Annotated tags used for development >>> Testing on Unseen data

The maximum entropy model described in the paper achieves 96.6% accuracy, correlating contextual information between development & testing texts instead of relying on the distributional assumptions.

The probability model takes “histories” and allowable tags together to compute correlated positive model parameters with respective features. In this case, the sequence of words and tags are fed as training data where the parameters are then chosen in a way to maximize the likelihood of the training data using p:

$$L(p) = \prod_{i=1}^{n} p(h_i, t_i) = \prod_{i=1}^{n} \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h_i,t_i)}$$

Essentially, the goal here is to maximize the entropy of a distribution subject to certain constraints. The entropy is defined as:

$$H(p) = -\sum_{h\in\mathcal{H},t\in\mathcal{T}} p(h,t)\log p(h,t)$$

Here, the constraint forces the model to match its feature expectations with those observed in the training data. This proves to be difficult as the set of tag contexts are very large & the model expectations can’t be computed directly. Therefore, an approximation function is used. The paper shows that if the distribution has the form :

$$p(h,t) = \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h,t)}$$

And satisfies the entropy constraints, it maximizes the entropy *H(p)* over distributions that satisfy the same constraints, and at the same time also maximizes the likelihood *L(p)*over other distributions of the same form.

Now, the features may activate on any word or tag in history 'h', and encode information that help predict tag 't'. If the features exist in the feature set of the model, the corresponding model parameter will serve as a weight for the corresponding contextual predictor towards the probability of observation a certain tag. Hence the features are generated for tagging using predefined templates where the conditions range from not rare or rare for all words. The model maintains low frequency of features that occur less than 10 times in the data.

Finally, during testing the test corpus is computed one sentence at a time to compute the tag sequence that generates highest probability.

Search Algorithm implied during model testing is as follows:

Let $W = \{w_1 \ldots w_n\}$ be a test sentence, and let s_{ij} be the jth highest probability tag sequence up to and including word w_i. The search is described below:

1. Generate tags for w_1, find top N, set s_{1j}, $1 \leq j \leq N$, accordingly.
2. Initialize $i = 2$
   (a) Initialize $j = 1$
   (b) Generate tags for w_i, given $s_{(i-1)j}$ as previous tag context, and append each tag to $s_{(i-1)j}$ to make a new sequence
   (c) $j = j + 1$, Repeat from (b) if $j \leq N$
3. Find N highest probability sequences generated by above loop, and set s_{ij}, $1 \leq j \leq N$, accordingly.
4. $i = i + 1$, Repeat from (a) if $i \leq n$
5. Return highest probability sequence, s_{n1}

References:

1. https://web.stanford.edu/~jurafsky/ws97/CL-dialog.pdf
2. https://nlp.stanford.edu/courses/cs224n/2015/reports/14.pdf
3. https://www.aclweb.org/anthology/P13-1129.pdf
4. https://web.stanford.edu/~jurafsky/slp3/24.pdf

5. https://pdfs.semanticscholar.org/df59/bf964b9f6db81dfbce3ad4c250dcca74e0b5.pdf?_ga=2.34841249.1267692646.1571767317-878656860.1571024233
6. https://github.com/dcavar/python-tutorial-for-ipython/blob/master/notebooks/Non-Deterministic%20Automaton%20Computing.ipynb