CSE 842: NATURAL LANGUAGE PROCESSING
HOMEWORK 2 ASSIGNMENT

Due Date: Nov. 3, 2019 at 11:59 PM

PROBLEM 1: NEURAL NETWORK FOR PART-OF-SPEECH TAGGING

Description: For this problem you will need to implement and evaluate a neural network (specifically, an RNN or LSTM) for POS tagging. Use any deep learning framework of your choice. I recommend trying Keras with scikit-learn. (Other frameworks are faster & allow more experimentation, but Keras is fast to understand.)

Tutorials: The following tutorials may be helpful in getting started.
PyTorch: Classifying Names with a Character-Level RNN
TensorFlow & Keras: RNN
Keras: Step by step
Keras: Multi-class classification

Code Requirements:

1. Neural net: Implement an RNN or LSTM using a deep learning frame work of your choice, sklearn, numpy, nltk, etc.
2. Cross-validation: Use an 80%-20% training-testing split for cross-validation.
   Optional (not required): use an 80-10-10 training-development-test split.
3. Features to test: capitalization, word position in sentence (first or last), word contains numbers & letters, word has a hyphen, entire word is capitalized, word is a number, first four characters of suffixes & prefixes (e.g., -ed, -ing, -ous), Glove embeddings, etc. You must incorporate at least 5 of these features. Using more will improve model performance.
4. Hyperparameters to test: number of hidden layers, optimizers (e.g., Adam vs. SGD), dropout rates, activation functions (sigmoid vs. ReLu vs. tanh), embeddings (dimension size, pre-trained vs. learned), learning rates, etc. You must test at least 3 different hyperparameters.
5. Dataset: Use the NLTK Treebank with the Universal tagset. This tagset has 12 classes: Noun, Verb, Pronoun, Adjective, Adverb, Adpositions, Conjunctions, Determiner, Cardinal Numbers, Particles, Other, Punctuation.
   ```
   tagged_sentences = nltk.corpus.treebank.tagged_sents(tagset='universal')
   ```

General implementation tips: Depending on your modeling framework, you might need to initialize the hidden layer with either zeroes or nonzero values. Features will need to be represented as vectors and DictVectorizer might be useful (depends on your processing). Sentences have different lengths, but you'll need to use a fixed length. There are different ways to handle this including: take a maximum length from the dataset, or take the average and make it a little longer. Once you choose a length, you need to do padding if the input sentence is too short, or some form of shortening if it's too long.

What to submit:

1. Your Python code implementing your *best* NN model, including feature representation, training, testing, and evaluation; name the file *yournetid_hw2p1.py*.
2. In the written report, include: how to run your code, features tested, hyperparameters tested. Using a table format, report the *accuracy* of your NN under the different features & hyperparameters.

PROBLEM 2: HIDDEN MARKOV MODEL (HMM)

Description: For this problem, you are to calculate the transition & emission probability matrices for an HMM. You do not have to code the full HMM or Viterbi decoding, just the calculations to get these two matrices.

Code Requirements:

1. Calculate the MLE for the transition probabilities (A) matrix of an HMM. Remember these are the probabilities that we would move from one state (POS tag) to another.
2. Calculate the MLE for the emission probabilities (B) matrix of an HMM. Remember these counts represent how often we see a specific word with a tag.
3. Dataset: use the news category of the Brown corpus, with the universal tagset.
   `brown.tagged_words(categories='news', tagset='universal')`

What to submit:

1. Your Python files implementing the calculation of transition (A) matrix & emission (B) matrix. Please name your Python file as *yournetid_hw2p2.py.*
2. In the written report, include how to run your code and a sample of the 2 matrices.
   a. For the A matrix, report the entire table (12 tags in rows & 12 tags in columns with transition probabilities in the cells).
   b. For the B matrix, report a subset of the calculations. Your table should have 12 rows, one for each POS tag. The columns should represent the emission probabilities for the following words: science, work, but, well, like, buffalo or a meaningful word of your choice (e.g., your name, home city/country, favorite color, etc.). Note: some words might not appear in the corpus subset though.

PROBLEM 3: PCFG

Production Rule	Probability
S → VP	1.0
VP → Verb NP	0.7
VP → Verb NP PP	0.3
NP → NP PP	0.3
NP → Det Noun	0.7
PP → Prep Noun	1.0
Det → the	0.1
Verb → cut | ask | …	0.1
Prep → with | in | …	0.1
Noun → envelope | person | scissors | …	0.1

Questions to answer:

1. A sentence can have more than one parse tree that is consistent with a CFG. How do non-probability-based CFGs and PCFGs differ in terms of handling parsing ambiguity?
2. Using the PCFG in the table above, draw the top-ranked parse tree for the sentences:
    a. Cut the envelope with scissors.
    b. Ask the person with scissors.
3. Semantically the parse tree of 2b is incorrect. Describe how you can lexicalize the PCFG to fix this, including how to modify the PCFG rules according to your lexicalization scheme.
4. Production rules might be assigned a probability of 0 if they are never seen in the training data. Describe one way to avoid this in a lexicalized PCFG.

SUBMISSION SUMMARY

Please submit the Python files that solve problems 1 & 2 and *one* written report in **PDF** format named *yournetid_hw2_report.pdf*. If you zip your files together, include your NetID in the filename.