Problem set 1: NLP I 100 points

How to approach this assignment

- You must use **Jupyter notebooks**. We'll run your code in ondemand or narnia. Let us know which one you used to test that your code works. Develop code directly in that environment to avoid deductions. We will not use Google colab.
- When you submit your code, include this statement with your submission: *"I have confirmed that my code is functional in the [ondemand|narnia] environment."*
- Start working **immediately** with your peer on PS 1. Don't underestimate time.
- Teams are **assigned** for solving. Each team member should engage with each problem and contribute to code and writing. You are not allowed to divide by problem. Work together.
- Use this workflow: Meet and discuss the solution for a problem. For a machine problem, jointly draft pseudocode (e.g., using a whiteboard).
- Submit a single code as **Jupyter notebooks per problem** in **Python 3.**
- **Comment** code in notebooks for facilitating grading. Deductions will be made for uncommented or sloppily commented code
- Code that does not run is awarded 0 points. Ensure you have time to test the team's code carefully at ondemand.rc.rit.edu. Adjust as needed before deadline.
- Each problem's code should be accompanied by a **PDF write-up**. You should use Overleaf and learn LaTeX and BibTex. You **may not stop at results – include results and discuss them critically.** Use equations, examples, tables, figures. Deductions are made if only reporting results without discussing them.
- Per problem, **report on who did what**. Ex: *Solving*-jointly, *Coding*-team member 1, *Debugging*-team member 2, *Analyzing*-team member 2, and *Writing*-jointly.
- In the writeups, use a reference section generated with BibTex in a consistent citation format (e.g. ACL, APA, IEEE) to cite sources in the write-up.
- Submit your team's report and notebook(s) as **ps1-LastName-LastName** Example: *ps1-alm-proano.[zip|tar.gz]* in the assigned Dropbox. Remember to include the statement from the second item above.
- Deductions will be made if you don't follow the submission instructions.

A. Implement a collocation finder. Implementation on your own. 20 pts.
(Acknowledgement: Based on Prud'hommeaux with modifications.)

You will write a program to identify collocations or words that occur near each other.

Prepare the text (fine to do outside your python program):

1. Download a decently sized corpus (between 500,000 and 2 million words) from Gutenberg.
2. Using regular expressions or other methods, write a function to remove any extraneous stuff from the text (start/end text from Gutenberg).

Analyze the text with PMI

3. Read in the text, then tokenize and lowercase it (e.g., use `nltk.word_tokenize` and string functions.
4. Calculate the unigram frequencies of the text. You may use NLTK, and its FreqDist data structure may be convenient.

5. Count 2-word adjacent and non-adjacent collocations in each sentence in your text using a 4-word sliding window and considering all pairs of words in the window.
6. Filter out the collocations that contain a stop word. You may use NLTK's stop list:
   ```
   from nltk.corpus import stopwords
   stoplist = stopwords.words('english')
   ```
   You are recommended to append to the stoplist to cover most English function words. Tell us which words you added in your report.
7. Calculate the **pointwise mutual information (PMI)** of every collocation with count greater than 2 with the following equation, which can be derived if assuming that the number of unigrams (*N* in the equation below) is approximately equal to the number of collocations.

$$PMI(w_1, w_2) = log_2 \left(N \frac{count(w_1, w_2)}{count(w_1) count(w_2)} \right)$$

In your report, please include and discuss:

1. What are 5 good collocations identified using PMI?
2. What are 5 not-so-good collocations identified using PMI?
3. Considering the result, what could be improved for an automated collocation identifier?
4. Describe a potential application for your collocation identification system.
5. Roughly a paragraph excerpt of your text should be included with your report and the full text should be uploaded with the code submission.
6. In the submission, include (i) the Jupyter notebook as `ps1.1BLastNames`, (ii) the text file from 5 as `ps1.1_LastNames.txt`, (iii) a PDF writeup as `ps1.1_LastNames.pdf`

2. Conduct a text classification experiment. Implementation on your own. 30 pts.

You will implement Yarowsky's basic log-likelihood decision list classifier and apply this algorithm to word sense disambiguation of homographs—words written but not usually pronounced the same way (consider text-to-speech). You will train on two training sets, respectively, and then use the corresponding test sets to evaluate your implementation.

The data are available in myCourses. It consists of two parts, each split into training (900 instances) and test (100 instances) sets. Each line in the file contains the target word's sense label ({*bass, *bass}* or {*sake, *sake}*), followed by a colon, followed by five word of left context, the target word, and five word of right context (i.e., a word window of length 11). Note that punctuation is not considered a word. The sense is indicated by including or excluding an asterisk next to the label. Interpret lines beginning or not beginning with an asterisk this way :

Set 1: ***bass** is the FISH sense; if no asterisk then it is MUSIC
Set 2: ***sake** is the BEER sense, if no asterisk then it is CAUSE

Follow these steps:

A. Read a classical paper: Yarowsky, D. *Decision lists for lexical ambiguity resolution: Application to Accent Restoration in Spanish and French*. While the use case was accent restoration, you will apply the technique to homograph disambiguation.
A. Preprocess the text: At least cast words to lowercase and remove punctuation.

Problem set 1: NLP I 100 points

B. Implement the algorithm with *features* such as the following:
Word within ±10 words context (minimally do this)*, word in position -k, Word in position k, Word in position -1, POS in position k, POS in position –k (for ±k = {1, 2})*, or your own creative ideas.
C. For each of the two ambiguous sense cases, compute the log-likelihood of the features based on training data statistics. Please see p. 91 in the paper for details.
    - Hint: We have talked about how to estimate the probability of bigrams, based on corpus counts that consider bigram and unigram counts. Similarly, consider that a particular sense occurs with certain features more often, and that a given feature also occurs with both senses.
    - Decide how to account for cases when a feature does not occur with both.
D. Rank-order the inferred decision rules you obtain into a decision list (DL).
E. With the test set, use the trained DL to classify test instances, based only on the highest ranked test (rule) in the DL, i.e., the most predictive decision rule.

Written reporting:

1. Summarize the task and assumptions made. What are other uses for a DL?
2. Explain what you did conceptually and any implementation decisions you took.
3. Report on the top-10 decision rules for BASS vs. SAKE. You may go down the DL.
4. Report on performance for each test set:
    - For each case, compare against a baseline that assigns the most likely sense found in the training set, to all instances in the test set for a given word. For *bass* vs *sake*: What is the baseline *sense label*? What is the result of this simple baseline on the *test set*? Use a table.
    - What improvement, if any, did you get over baseline? Report on accuracy.
    - Include confusion matrices for BASS and SAKE and your observations.
    - Include 3 example instances from each test set that were correctly classified vs. 3 incorrectly classified, per case.
    - Compute these metrics per case and include them in the table. Explain how you operationalized the chosen metrics.
        1. Compute % absolute change in accuracy compared to the baseline.
        2. Compute % error reduction over baseline.
        3. Compute macro-averaged vs micro-averaged precision and recall.
5. Provide a written discussion where you reflect upon results and reason about them. For example: *What influenced the results on performance metrics*? *What do the confusion matrices tell you?* For examples provided, *what went right/wrong*? If you encountered any concerns along the way, mention those too.
6. We will run the code. Describe how to update train and test sets in the notebook.
7. Submit Juputer notebook `ps1.2_Lastnames` and writeup `ps1.2_LastNames.pdf`

Bonus point: 7 extra pts.: Explore the impact on performance if varying the size of the training data, plot a learning curve, and discuss results.

3. Compare text generated with an n-gram model vs. with a fine-tuned LLM. An implementation may consult external resources, including code. 30 pts.

Problem set 1: NLP I 100 points

First – train a bigram model and generate text with it.

1. Create a new corpus from Gutenberg, combining multiple texts from an author or one topic into a superdoc, after you strip the Gutenberg pre/post matter legal texts etc. using your text pre-processing function from problem 1 above.
2. Compute the `bigrams` using NLTK.
3. Prepare a bigram grammar. You may use NLTK's `FreqDist` to count bigrams and `ConditionalFreqDist` to determine the likelihood of a word given other words.
4. Prepare a function to generate text.
5. Prompt the model with some selected starting words and generate texts of 50 words.
6. Explore distinct words (e.g., a common word at the start of sentences, words specific to the author or topic, function vs content words). Discuss 3 cases and include the 50 words generated in textboxes.

Extra credit: Compare with *character bigrams* or *word trigrams*, for 5 bonus points.

Second – fine-tune a GPT model and generate text with it.

7. Use Hugging Face's `transformers` to fine-tune a prior GPT with this corpus.
8. If you are new to this library, it is fine to consult resources (including ChatGPT and example scripts) about preparing code to fine-tune it.
9. Prompt the model with the same starting words. Again, discuss 3 examples, with generated text included.
10. Expand with longer prompts and contrast output. Discuss.

Submit the report as `ps1.3_LMvsLLM_LastNames.pdf` and the Jupyter notebook as `ps1.3_LMvsLLM_LastNames.`

Extra credit (up to 10 pts): Compute BLEU and/or perplexity and discuss.

4. Exploring the ChatGPT developer framework. 10 points.

Using the Chat-GPT developer API sandbox, pick a problem. It is fine to pick a canned example but modify it in a creative way. Report in `ps1.4_ChatGPTAPI_LastNames.pdf` on (a) what you opted for exploring, (b) what adaptations you made, (c) your findings, (d) limitations observed. Include prompts used, and any code as `ps1.4_ChatGPTAPI_LastNames` if applicable.

5. Pair-written essay. 10 points.

A. For this essay, you will practice co-authoring with your peer in Overleaf.
B. Choose a published paper from the ACL Anthology (aclweb.org/anthology) that considers some of the ethical issues LLMs grapple with.
C. Review the CORE Conference Portal (http://portal.core.edu.au/conf-ranks/) to see if the paper's venue was ranked and in what category. State it as a comment at the end.
D. Include a title for your essay, and the names of yourselves as authors.
E. Critically discuss the paper **jointly** and comment on links between CL/NLP and other disciplines. Length: 1 p. with a standardly formatted *References* section using BibTeX (a BibTeX entry is available in a paper's ACL Anthology entry).
F. Include as a separate PDF if you package entitled `1.5_Essay_LastNames.pdf`.