Assignment C

CSE 517: Natural Language Processing - University of Washington

Winter 2023

Please consult the course website for current information on the due date, the late policy, and any data you need to download for this assignment. This assignment is designed to advance your understanding of some ethical challenges that arise in NLP and of sequence models.

Submit: You will submit your writeup (a pdf), your code (do not include data), and the decoded text from problem 3, via Gradescope. Instructions can be found here. Note that you will make three submissions: one for the pdf, one for the code, and one for the decoded text.

1 Computational Ethics

This assignment is based on homework #3 from the course "Computational Ethics for NLP" at CMU, taught by Yulia Tsvetkov and Alan Black.

Goals Online data has become an essential source of training data for natural language processing and machine learning tools; however, the use of this type of data has raised concerns about privacy. Furthermore, the detection of demographic characteristics is a common component of micro-targeting. In this assignment, you will explore how to obfuscate demographic traits, specifically gender. The primary goals are (1) develop a method for obfuscating an author's gender and (2) explore the trade-off between obfuscating an author's identity and preserving useful information in the data.

Overview The primary dataset we provide consists of posts from Reddit. Each post is annotated with the gender of the post's author ("op_gender") and the subreddit where the post was made ("subreddit"). The main text of the post is in the column "post_text". The contents of the provided data include:

- `classify.py`: a classifier that predicts the author's gender and the subreddit for a post (example run: `python classify.py --test_file dataset.csv`). Note that this file also uses the two provided pickle files.
- `dataset.csv`: your primary data
- `background.csv`: additional Reddit posts that you may optionally use for training an obfuscation model. We will also provide a larger version.
- `female.txt`: a list of words commonly used by women
- `male.txt`: a list of words commonly used by men

We note that this assignment uses a simple operationalization of gender as binary; we will return to this point below.

The provided classifier achieves an accuracy around 65% at identifying the gender of the poster and an accuracy around 85% at identifying a post's subreddit when tested over `dataset.csv`. Your goal in this assignment is to obfuscate the data in dataset.csv so that the provided classifier is unable to determine the gender of authors, while still being able to determine the subreddit of the post. Note that in this set-up, we treat the provided classifier as a fixed, blackbox adversary (please do not try to hack it). This assignment was largely inspired by Reddy and Knight [5], which may be a useful reference. Scenarios where this obfuscation model might be useful could be social media users who want to preserve their privacy by hiding their gender, without losing the meaning of their post. You could also imagine this is a dataset of health records or other sensitive information that needs to be anonymized before providing it to researchers.

Requirements First, build a baseline obfuscation model:

- For each post in `dataset.csv`, if the post was written by a man ("M") and it contains words from `male.txt`, replace these words with a random word from `female.txt`.
- Obfuscate posts written by women ("W") in the same way (i.e., by replacing words from `female.txt` with random words from `male.txt`).
- Test `classify.py` on your obfuscated data and report what happens to the two accuracy measurements discussed above.

Second, improve your obfuscation model:

- Instead of replacing words from `male.txt` with randomly chosen words from `female.txt`, choose a semantically similar word from `female.txt`. Do the same in reverse. You may use any metric you like for identifying semantically similar words, but you should explain why you chose it. We recommend starting with cosine distance between pretrained word embeddings (available, for example, here)
- Test `classify.py` on data obfuscated using your improved model and analyze the results. The classifier should perform close to random at identifying gender and should obtain at least 79% accuracy on classifying the subreddit.

Third, experiment with some basic modifications to your obfuscation models. For example, what if you randomly decide whether or not to replace words instead of replacing every lexicon word? What if you only replace words that have semantically similar enough counterparts?

Extra Credit: Advanced Obfuscation Develop your own obfuscation model. We provide `background.csv`, a large dataset of Reddit posts tagged with gender and subreddit information that you may use to train your obfuscation model. We also provide a larger version of the background corpus. Your goal should be to obfuscate text so that the classifier is unable to determine the gender of an author (no better than random guessing) without compromising the accuracy of the subreddit classification task. However, creative or thorough approaches will receive full credit, even if they do not significantly improve results. Some ideas you may consider:

- Develop your own lexicons using pointwise mutual information scores or log odds with a Dirichlet prior; see, e.g., Jurafsky et al. [2]

- Follow the procedure described by Reddy and Knight [5]
- Use an adversarial objective as described by Pryzant et al. [4] to train a model that is good at predicting subreddit classification but bad at predicting gender. The key idea in this approach is to design a model that does not encode information about protected attributes (in this case, gender).
- Use a model for style transfer, such as in Prabhumoye et al. [3]

In your report, include a description of your model and results, and clearly label it "Extra Credit."

Deliverables Submit a report that is no more than three pages long. It should include:

- A scatterplot where subreddit accuracy on `dataset.csv` is plotted on the x-axis, gender accuracy on the same data is plotted on the y-axis, and every model you built is a (labeled) point.
- The same information presented in a table (each row a system, one column for subreddit accuracy and one column for gender accuracy).
- One paragraph to describe each system you considered.
- Qualitative examples of text obfuscated with your models.
- As noted, this assignment assumes gender can be treated as a binary variable. Reflect briefly (a paragraph or so) on alternative ways one could formulate this problem, and the pros and cons of doing so.
- Discuss (in a paragraph or so) the ethical implications of obfuscation more generally, drawing from concepts you encountered in class, readings, or elsewhere.

2 Reading and Reflection

Read Hovy and Spruit [1]; write a brief summary of the paper's argument (one paragaph) and then write a personal response (one paragraph).

3 Sequence Decoding

We provide to you two files:

- `15pctmasked.txt` contains single sentences, one per line (i.e., newline characters delimit sentences), each with some of its characters "masked"; characters are delimited by whitespace, masked characters are replaced with "<mask>", spaces are denoted "<s>", and the end-of-sequence symbol is "<eos>"
- `lm.txt` contains a bigram language model over characters; the format on each line is "a space b tab $p(b \mid a)$" with special space and end-of-sequence characters denoted as above.

Your job is to decode the sentences, replacing each "masked" character with a character from the bigram model's vocabulary. For input $\boldsymbol{x}$, your output $\boldsymbol{y}$ should be the sequence that maximizes $p(\boldsymbol{y})$ under the bigram language model, subject to the constraint that, for every position i where $\boldsymbol{x}$ is *not* masked, $y_i = x_i$. (In plain English, we want you to find the most likely unmasked sequence consistent with the input, under the model we gave you.) Hint: you should adapt the Viterbi algorithm to solve this problem exactly in polynomial time and space.

Your output file should be in the same format as the input, but with every mask token replaced as described above. Your writeup should explain how you generate a score for each possible choice and how you recover the best sequence. You may use existing libraries and tools for this problem (be sure to acknowledge them in your solution), but we do not expect they will be very helpful.

4 American National FSA – Eisenstein 9.3 (p. 213)

Construct (i.e., draw on the page; a TA recommends this tool) a finite-state automaton in the style of figures 9.2 and 9.3 (pp. 187–8), which handles the following examples:

- *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
- *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N

Be sure that your FSA does not accept any further derivations, such as **nationalizeral* and **Americanizern*.

5 Viterbi for Second-Order Models

At the end of the CRF lecture (slide 106), we proposed a second-order sequence model that defines scores of *triplets* of labels, $s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2})$. The problem to be solved when we decode with this model is:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^\ell} \sum_{i=0}^{\ell-1} s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2}) \tag{1}$$

Note that there are ℓ terms in the sum. y_0 (which is a special START label) appears only in the first term ($i = 0$) and $y_{\ell+1}$ (which is a special STOP label) appears only in the last term ($i = \ell - 1$). In a conventional trigram-style model, we might also have "$s(\boldsymbol{x}, -1, y_{-1} = \text{START}, y_0 = \text{START}, y_1)$," but including such a term doesn't add any expressive power to what is written above.

The Viterbi algorithm can be adapted for this model. The form of the algorithm is nearly identical to what we saw in lecture. First, prefix scores are computed "left to right," each using a formula with a local max, and along with each prefix score we also store a backpointer that stores the "argmax." Then the backpointers are followed "right to left" to select a label for each word in turn. Here is most of the algorithm:
Input: scores $s(\boldsymbol{x}, i, y'', y', y), \forall i \in \langle 0, \ldots, \ell \rangle, \forall y'' \in \mathcal{L}, \forall y' \in \mathcal{L}, \forall y \in \mathcal{L}$
Output: $\hat{\boldsymbol{y}}$

1. Base case: $\heartsuit_2(y', y) = s(\boldsymbol{x}, 0, \text{START}, y', y)$
   (Note that the subscript in "$\heartsuit_2$" refers to the last position in the triple, here, the word labeled "y," while the "0" argument to s refers to the first position in the triple.)
2. For $i \in \langle 3, \ldots, \ell - 1 \rangle$:
   - Solve for $\heartsuit_i(*, *)$ and $b_i(*, *)$.
3. Special case for the end:

$$\heartsuit_\ell(y', y) = s(\boldsymbol{x}, \ell - 1, y', y, \text{STOP}) + \underbrace{\max_{y'' \in \mathcal{L}} s(\boldsymbol{x}, \ell - 2, y'', y', y) + \heartsuit_{\ell-1}(y'', y')}_{b_\ell(y', y) \text{ is the "argmax"}}$$

   The above looks a little different from what you saw in lecture, where we calculated "$\heartsuit_{\ell+1}(\text{STOP})$." Writing it as we've done here is just a slightly different choice that gives you a little more of a hint. To see why, take a close look at slide 70.

4. $(\hat{y}_{\ell-1}, \hat{y}_\ell) \leftarrow \underset{y', y \in \mathcal{L}^2}{\operatorname{argmax}} \heartsuit_\ell(y', y)$

5. For $i \in \langle \ell - 2, \ldots, 1 \rangle$:

   - $\hat{y}_i \leftarrow b_{i+2}(\hat{y}_{i+1}, \hat{y}_{i+2})$

Deliverables:

1. Give the recurrence for $\heartsuit_i(y', y)$ and for $b_i(y', y)$ (the boxed bit of the algorithm above). We've given you the base case and the special case for the end of the sequence.

2. Analyze the runtime and space requirements of this algorithm as functions of $|\mathcal{L}|$ (the number of labels) and ℓ (the length of the input sequence $\boldsymbol{x}$ and the output sequence $\hat{\boldsymbol{y}}$).

6 Extra Credit: Introducing PCFGs

In this problem, you'll become acquainted with **probabilistic context-free grammars** (PCFGs), a more powerful family of models that can be used to parse sequences of words into trees. Chapters 9 and 10 in the textbook give extensive motivation for these kinds of models, as well as algorithms for their use. This problem will reveal a connection between PCFGs and HMMs.

A PCFG is defined as:

- A finite set of "nonterminal" symbols $\mathcal{N}$
  - A start symbol $S \in \mathcal{N}$
- A finite alphabet Σ, called "terminal" symbols, distinct from $\mathcal{N}$
- Production rule set $\mathcal{R}$, each of the form "$N \rightarrow \boldsymbol{\alpha}$" where
  - The lefthand side N is a nonterminal from $\mathcal{N}$
  - The righthand side $\boldsymbol{\alpha}$ is a sequence of zero or more terminals and/or nonterminals: $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$
    - Special case: **Chomsky normal form** constrains $\boldsymbol{\alpha}$ to be either a single terminal symbol or two nonterminals
- For each $N \in \mathcal{N}$, a probability distribution over the rules where N is the lefthand side, $p(* \mid N)$. We will use p_r to denote the conditional probability corresponding to rule $r \in \mathcal{R}$.

A PCFG defines the probability of a tree $\boldsymbol{t}$ as:

$$p(\boldsymbol{t}) = \prod_{r \in \mathcal{R}} p_r^{\text{count}_{\boldsymbol{t}}(r)} \tag{2}$$

where $\text{count}_{\boldsymbol{t}}(r)$ is the number of times rule r is used in the tree $\boldsymbol{t}$.

The definition of an HMM was given in class; we spell it out more carefully here.

- A finite set of states $\mathcal{L}$ (often referred to as "labels")
  - A probability distribution over initial states, p_{start}
  - One state denoted ⬡ is the special stopping state
- A finite set of observable symbols $\mathcal{V}$

- For each state $y \in \mathcal{L}$:
  - An "emission" probability distribution over $\mathcal{V}$, $p_{emission}$
  - A "transition" probability distribution over next states, $p_{transition}$

An HMM defines the probability distribution of a state sequence $\boldsymbol{y}$ and an emission sequence $\boldsymbol{x}$ (length n, excluding the stop symbol) as shown on slide 36 in the lecture.

1. Demonstrate how to implement any HMM using a PCFG. Do this by showing formally how to construct each element of the PCFG ($\mathcal{N}$, S, Σ, $\mathcal{R}$, and p) from the HMM. Hint: think of a state sequence as a right-branching tree; the nonterminals will correspond to HMM states/labels, and you will probably want to add a new nonterminal to serve as the start state S.

2. Transform the grammar you defined above into Chomsky normal form. Every tree derived by the original grammar must have a corresponding tree in the new grammar, with the same probability. Hint: his might be easier if you add a special "start" word to the beginning of every sequence recognized by the new grammar, and you will likely need to add some new nonterminals, as well. On page 202 of your textbook, the transformation is sketched out for arbitrary context-free grammars that are not probabilistic; to fully complete this exercise, you need to handle the probabilities as well as the rules, but only need to show how to do it for your HMM-derived PCFG. Try to give a more precise definition of the transformation than is given in the textbook.

3. Explain how to transform your new PCFG's derivations back into derivations under the original PCFG.

4. Is it practical to use the CKY algorithm to find the most probable state sequence for a sentence $\boldsymbol{x}$, under an HMM? Why or why not?

7 Extra Credit (Challenging): Unseen Bigrams – Based on Eisenstein 6.4 (p. 135)

Consider a simple language in which each token is drawn from the vocabulary $\mathcal{V}$ with probability $\frac{1}{|\mathcal{V}|}$, independent of all other tokens.

Given a corpus with M tokens, what is the expectation of the fraction of all possible bigrams that have zero count (i.e., did not appear in the corpus)? Find a tight upper bound of the expectation. Show all your steps.

Example Suppose $\mathcal{V} = \{a, b, c\}$, $M = 5$, corpus $= accba$.

There are 4 bigrams in the corpus: ac, cc, cb, ba. And a total of $|\mathcal{V}|^2 = 3 \times 3 = 9$ possible bigrams.

References

[1] Dirk Hovy and Shannon L. Spruit. The social impact of natural language processing. In *Proc. of ACL*, 2016. URL https://aclanthology.org/P16-2096.

[2] Dan Jurafsky, Victor Chahuneau, Bryan R. Routledge, and Noah A. Smith. Narrative framing of consumer sentiment in online restaurant reviews. *First Monday*, 19(4), April 2014. URL https://firstmonday.org/ojs/index.php/fm/article/view/4944.

[3] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W. Black. Style transfer through back-translation. In *Proc. of ACL*, 2018. URL https://aclanthology.org/P18-1080.

[4] Reid Pryzant, Young joo Chung, and Dan Jurafsky. Predicting sales from the language of product descriptions. In *Proc. of eCOM@SIGIR*, 2017. URL http://ceur-ws.org/Vol-2311/paper_3.pdf.

[5] Sravana Reddy and Kevin Knight. Obfuscating gender in social media writing. In *Proceedings of the First Workshop on NLP and Computational Social Science*, 2016. URL https://aclanthology.org/W16-5603.