CSCI-GA 2590: Natural Language Processing, Spring 2023

Representing and Classifying Text

Name

NYU ID

Collaborators:

By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Welcome to your first assignment! The goal of this assignment is to get familiar with basic text classification models and explore word vectors using basic linear algebra tools. **Before you get started, please read the Submission section thoroughly**.

Submission

Submission is done on Gradescope.

Written: When submitting the written parts, make sure to select **all** the pages that contain part of your answer for that problem, or else you will not get credit. You can either directly type your solution between the `shaded` environments in the released `.tex` file, or write your solution using a pen or stylus. A `.pdf` file must be submitted.

Programming: Questions marked with "coding" next to the assigned to the points require a coding part in `submission.py`. Submit `submission.py` and we will run an autograder on Gradescope. You can use functions in `util.py`. However, please do not import additional libraries (e.g. `sklearn`) that aren't mentioned in the assignment, otherwise the grader may crash and no credit will be given. You can run `test_classification.py` and `test_embedding.py` to test your code but you don't need to submit it.

## Problem 1: Natural language inference

In this problem, you will build a logistic regression model for textual entailment. Given a *premise* sentence, and a *hypothesis* sentence, we would like to predict whether the hypothesis is *entailed* by the premise, i.e. if the premise is true, then the hypothesis must be true.
Example:

label	premise	hypothesis
entailment	The kids are playing in the park	The kids are playing
non-entailment	The kids are playing in the park	The kids are happy

1. [1 point] Given a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ where $y \in \{0, 1\}$, let ϕ be the feature extractor and w be the weight vector. Write the maximum log-likelihood objective as a *minimization* problem.

2. [3 point, coding] We first need to decide the features to represent x. Implement `extract_unigram_features` which returns a BoW feature vector for the premise and the hypothesis.
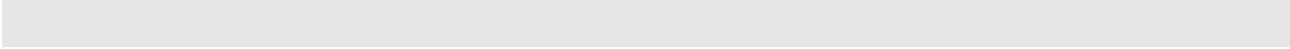
3. [2 point] Let $\ell(w)$ be the objective you obtained above. Compute the gradient of $\ell_i(w)$ given a single example $(x^{(i)}, y^{(i)})$. Note that $\ell(w) = \sum_{i=1}^{n} \ell_i(w)$. You can use $f_w(x) = \frac{1}{1+e^{-w \cdot \phi(x)}}$ to simplify the expression.
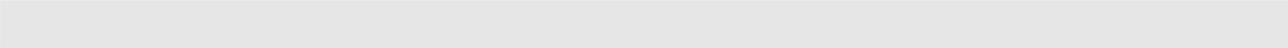
4. [5 points, coding] Use the gradient you derived above to implement `learn_predictor`. You must obtain an error rate less than 0.3 on the training set and less than 0.4 on the test set to get full credit *using the unigram feature extractor.*
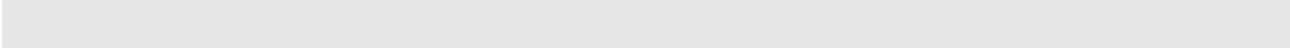
5. [3 points] Discuss what are the potential problems with the unigram feature extractor and describe your design of a better feature extractor.
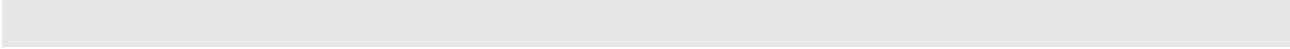
6. [3 points, coding] Implement your feature extractor in `extract_custom_features`. You must get a lower error rate on the dev set than what you got using the unigram feature extractor to get full credits.
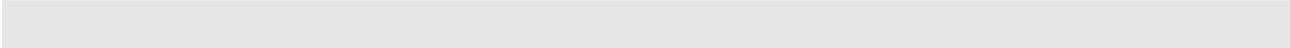
7. [3 points] When you run the tests in `test_classification.py`, it will output a file `error_analysis.txt`. (You may want to take a look at the `error_analysis` function in `util.py`). Select five examples misclassified by the classifier using custom features. For each example, briefly state your intuition for why the classifier got it wrong, and what information about the example will be needed to get it right.
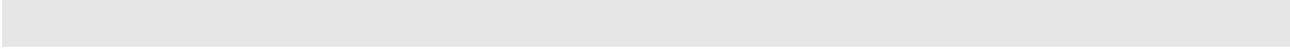
8. [3 points] Change `extract_unigram_features` such that it only extracts features from the hypothesis. How does it affect the accuracy? Is this what you expected? If yes, explain why. If not, give some hypothesis for why this is happening. Don't forget to change the function back before your submit. You do not need to submit your code for this part.
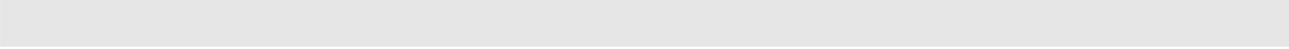
Problem 2: Word vectors

In this problem, you will implement functions to compute dense word vectors from a word co-occurrence matrix using SVD, and explore similarities between words. You will be using python packages `nltk` and `numpy` for this problem.

We will estimate word vectors using the corpus *Emma* by Jane Austen from `nltk`. Take a look at the function `read_corpus` in *util.py* which downloads the corpus.

1. [3 points, coding] First, let's construct the word co-occurrence matrix. Implement the function `count_cooccur_matrix` using a window size of 4 (i.e. considering 4 words before and 4 words after the center word).
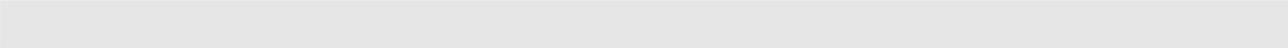
2. [1 points] Next, let's perform dimensionality reduction on the co-occurrence matrix to obtain dense word vectors. You will implement truncated SVD using the `numpy.linalg.svd` function in the next part. Read its documentation (`https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html`) carefully. Can we set `hermitian` to `True` to speed up the computation? Explain your answer in one sentence.
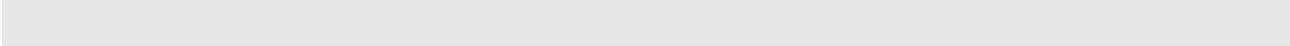
3. [3 points, coding] Now, implement the `cooccur_to_embedding` function that returns word embeddings based on truncated SVD.

4. [2 points, coding] Let's play with the word embeddings and see what they capture. In particular, we will find the most similar words to a given word. In order to do that, we need to define a similarity function between two word vectors. Dot product is one such metric. Implement it in `top_k_similar` (where `metric='dot'`).
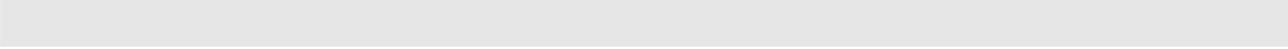
5. [1 points] Now, run `test_embedding.py` to get the top-k words. What's your observation? Explain why that is the case.

6. [2 points, coding] To fix the issue, implement the cosine similarity function in `top_k_similar` (where `metric='cosine'`).
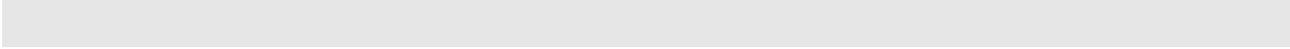
7. [1 points] Among the given word list, take a look at the top-k similar words of "man" and "woman", in particular the adjectives. How do they differ? Explain what makes sense and what is surprising.

8. [1 points] Among the given word list, take a look at the top-k similar words of "happy" and "sad". Do they contain mostly synonyms, or antonyms, or both? What do you expect and why?