CSCI-GA 2590: Natural Language Processing, Spring 2023

Representing and Classifying Text

Chuanyang Jin
cj2133

Feb 10

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.

$$\text{minimize} \; -\sum_{i=1}^{n} \Big[y^{(i)} \cdot \log(f_w(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - f_w(x^{(i)})) \Big]$$

$$\text{where } f_w(x^{(i)}) = \frac{1}{1 + e^{-w \cdot \phi(x^{(i)})}}$$

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.

Implemented.

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.

$$\ell(w) = -\sum_{i=1}^{n}\Big[y^{(i)} \cdot \log(f_w(x^{(i)})) + (1-y^{(i)}) \cdot \log(1-f_w(x^{(i)}))\Big]$$

$$\ell_i(w) = -y^{(i)} \cdot \log(f_w(x^{(i)})) - (1-y^{(i)}) \cdot \log(1-f_w(x^{(i)}))$$

$$\begin{aligned}
\frac{\partial \ell_i(w)}{\partial w} &= -\frac{y^{(i)}}{f_w(x^{(i)})} \cdot \frac{\partial f_w(x^{(i)})}{\partial w} + \frac{1-y^{(i)}}{1-f_w(x^{(i)})} \cdot \frac{\partial f_w(x^{(i)})}{\partial w} \\
&= -\frac{y^{(i)}}{f_w(x^{(i)})} \cdot f_w(x^{(i)}) \cdot (1-f_w(x^{(i)})) \cdot \phi(x^{(i)}) + \frac{1-y^{(i)}}{1-f_w(x^{(i)})} \cdot f_w(x^{(i)}) \cdot (1-f_w(x^{(i)})) \cdot \phi(x^{(i)}) \\
&= -y^{(i)} \cdot (1-f_w(x^{(i)})) \cdot \phi(x^{(i)}) + (1-y^{(i)}) \cdot f_w(x^{(i)}) \cdot \phi(x^{(i)}) \\
&= (f_w(x^{(i)}) - y^{(i)}) \cdot \phi(x^{(i)})
\end{aligned}$$

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*.

Implemented.

5. [3 points] Discuss what are the potential problems with the unigram feature extractor and describe your design of a better feature extractor.

The potential problems with the unigram feature extractor is that it does not distinguish between the premise and hypothesis. Whether a word appears in the premise or hypothesis makes no difference to the feature.

My design of a better feature extractor solves this problem. It keeps a dictionary of the words that appear only in the hypothesis but not in the premise, viewing them as a potential feature of "non-entailment". It also considers other words that appear in both sentences. As a result, it reduces the error rate on the test set from 0.36 to 0.24.

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.

Implemented.

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.

P: A cabin shot of a very crowded airplane
H: The airplane is quite crowded .
label=1, predicted=0, wrong
Intuition: Three words appear only in the hypothesis but not in the premise, producing an illusion of "non-entailment". The information that "quite" is actually a synonym of "very" will be needed to get rid of the confusion.

P: A building that portrays beautiful architecture stands in the sunlight as somebody on a bike passes by .
H: A bicyclist passes an esthetically beautiful building on a sunny day
label=1, predicted=0, wrong
Intuition: The hypothesis and the premise talk about the same thing in a quite different way. A more clear relation between "sunny day" and "sunlight", and "bicyclist" and "bike" will be needed.

P: Two guys are wearing uniforms who are running in the grass .
H: Two guys in uniforms are playing a sport on the grass
label=0, predicted=1, wrong
Intuition: Most words are the same in the hypothesis and the premise, and the classifier fails to distinguish the slight difference between "sport" and "running". More emphasis on this vital difference will be needed.

P: Two men in uniforms are in soccer field behind a line waiting to play .
H: two men waiting to play soccer in May
label=0, predicted=1, wrong
Intuition: The hypothesis has an extra information "in May". The classifier fails to recognize the importance of this extra information. More emphasis on this kind of extra information will be needed.

P: A woman in a purple dress talks on her cellphone and a man reads a book as a two-story bus passes by outside their window .
H: a woman calls her kids
label=0, predicted=1, wrong
Intuition: There's too much background information in the premise, which confuses the classifier. Eliminating the effect of useless information (lowering their weights) is needed.

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.

It improves the accuracy on the test set from 0.64 to 0.72. This is not what I expected.

To explain this, a hypothesis is that the information in the hypotheses is more important for the prediction. Using only the words in them can give considerably good prediction. More information in the premises will disturb the prediction. Another possibility is that the model implicitly learns how to predict the premise given the hypothesis even though it is not expected to do so.

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).

Implemented.

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.

Yes, since being neighbors is a mutual relationship for two words, our co-occurrence matrix will be Hermitian, and setting `hermitian` to `True` will speed up the computation.

3. [3 points, coding] Now, implement the `cooccur_to_embedding` function that returns word embeddings based on truncated SVD.

Implemented.

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'`).

Implemented.

5. [1 points] Now, run `test_embedding.py` to get the top-k words. What's your observation? Explain why that is the case.

Some common words frequently appear in the top-k words. Their representation vectors may have large entries. Even though the angles between their representation vectors and other vectors are large, they are likely to have large dot product outputs without normalization.

6. [2 points, coding] To fix the issue, implement the cosine similarity function in `top_k_similar` (where `metric='cosine'`).

Implemented.

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.

top k most similar words to man:
woman lady farmer charming _woman_ pert gallant ladies weak fine
top k most similar words to woman:
man charming farmer pert lady girl blush fine minute sweet

The difference is that “man” is more similar to “gallant” and “weak”, and “woman” is more similar to “blush”, “minute” and “sweet”.
It makes sense that “man” is similar to “gallant” and “woman” is similar to “blush” and “sweet”, since our model may learn some gender stereotypes from the texts.
It is surprising that “man” is also similar to “weak”, since we don’t usually associate men with weakness.

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?

They contain both synonyms and antonyms. I expect them to be synonyms since they are suppose to be similar words. In fact, however, some antonyms may also have close embeddings.