DS-GA-1011: Natural Language Processing with Representation Learning, Fall 2024

Representing and Classifying Text

Sophie Juco
smj490

Please write down any collaborators, AI tools (ChatGPT, Copliot, codex, etc.), and external resources you used for this assignment here.
Collaborators: None
AI tools: None
Resources: Notes from class lectures and previous classes' lectures (for some math proof questions); GeeksforGeeks and Stack Overflow (for some coding questions)

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**.

Due Date - 11:59 PM 09/20/2024

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: Naive Bayes classifier

In this problem, we will study the *decision boundary* of multinomial Naive Bayes model for binary text classification. The decision boundary is often specificed as the level set of a function: $\{x \in \mathcal{X} : h(x) = 0\}$, where x for which $h(x) > 0$ is in the positive class and x for which $h(x) < 0$ is in the negative class.

1. [2 points] Give an expression of $h(x)$ for the Naive Bayes model $p_\theta(y \mid x)$, where θ denotes the parameters of the model.

   Solution. **1** Since binary classification has two classes: positive (y=1) and negative (y=0), and the decision rule for Naive Bayes is to choose the class with the highest posterior probability, we can use log-odds ratio to represent Naive Bayes for our binary text classification:
   $$\begin{cases} \log(\frac{p_\theta(y=1|x)}{p_\theta(y=0|x)}) > 0 & ,y=1 \\ \text{Else} & ,y=0 \end{cases}$$

   2 So:
   $h(x) = \log(\frac{p_\theta(y=1|x)}{p_\theta(y=0|x)})$

   3 Expand with Bayes theorem:
   $h(x) = \log(\frac{p_\theta(x|y=1)p_\theta(y=1)}{p_\theta(x|y=0)p_\theta(y=0)})$

   4 With conditional independence assumption:
   $p_\theta(x|y) = \Pi_{i=1}^n p_\theta(x_i|y)^{x_i}$

   5 Sub $p_\theta(x|y)$ into function of $h(x)$:
   $h(x) = \log(\frac{\Pi_{i=1}^n p_\theta(x_i|y=1)^{x_i} p_\theta(y=1)}{\Pi_{i=1}^n p_\theta(x_i|y=0)^{x_i} p_\theta(y=0)})$

   6 Solve:
   $h(x) = \sum_{i=1}^n x_i \log(\frac{p_\theta(x_i|y=1)}{p_\theta(x_i|y=0)}) + \log(\frac{p_\theta(y=1)}{p_\theta(y=0)})$

   7 Let $\theta_{i,1} = p(x_i|y=1)$ and $\theta_{i,0} = p(x_i|y=0)$, rewrite:
   $h(x) = \sum_{i=1}^n x_i \log(\frac{\theta_{i,1}}{\theta_{i,0}}) + \log(\frac{p_\theta(y=1)}{p_\theta(y=0)})$ □

2. [3 points] Recall that for multinomial Naive Bayes, we have the input $X = (X_1, \ldots, X_n)$ where n is the number of words in an example. In general, n changes with each example but we can ignore that for now. We assume that $X_i \mid Y = y \sim \text{Categorical}(\theta_{w_1,y}, \ldots \theta_{w_m,y})$ where $Y \in \{0, 1\}$, $w_i \in \mathcal{V}$, and $m = |\mathcal{V}|$ is the vocabulary size. Further, $Y \sim \text{Bernoulli}(\theta_1)$. Show that the multinomial Naive Bayes model has a linear decision boundary, i.e. show that $h(x)$ can be written in the form $w \cdot x + b = 0$. [**RECALL:** The categorical distribution is a multinomial distribution with one trial. Its PMF is

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} \theta_i^{x_i} ,$$

where $x_i = \mathbb{1}[x = i]$, $\sum_{i=1}^{m} x_i = 1$, and $\sum_{i=1}^{m} \theta_i = 1$.]

Solution. **1** Using the solution to Problem 1.1, we can sub w_j in for θ_i since θ_w is the specific vocabulary we are working with:
$h(x) = \sum_{j=1}^{m} x_j \log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}}) + \log(\frac{p_\theta(y=1)}{p_\theta(y=0)})$

2 $\log(\frac{p_\theta(y=1)}{p_\theta(y=0)})$ is a constant so it represents b in the linear equation.
So, $h(x) = \sum_{j=1}^{m} x_j \log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}}) + b$

3 $\sum_{j=1}^{m} x_j \log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}})$ is a linear combination of x where $\log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}})$ is the weight of each x (or word).
So, $\log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}}) = w$ and $x = x$
where $\sum_{j=1}^{m} x_j \log(\frac{\theta_{w_j,1}}{\theta_{w_j,0}}) = w \cdot x$

4 So, $h(x) = w \cdot x + b$ □

3. [2 points] In the above model, X_i represents a single word, i.e. it's a unigram model. Think of an example in text classification where the Naive Bayes assumption might be violated. How would you allieviate the problem?

   Solution. **1** An example of where Naive Bayes assumption is violated is negative terms like "not good" VS "good". For example, "The dish is good" VS "The dish is not good". Under the Naive Bayes assumption, "not" and "good" are considered independently when in reality, the meaning of "good" is dependent on "not" preceding it or not.

   2 One way to solve this is using an n-gram model which considers groups of words rather than individual words and would thus pick up the "not" preceding the word "good". And, a properly trained model will understand that the preceding "not" negates the meaning of "good". □

4. [2 points] Since the decision boundary is linear, the Naive Bayes model works well if the data is linearly separable. Discuss ways to make text data works in this setting, i.e. make the data more linearly separable and its influence on model generalization.

Solution. **1** One way to make the data more linearly separable is with TF-IDF which transforms word counts into frequency scores. This gives more weight to words that are unique to a document and less weight to words that are common across all documents. This improves the generalization by focusing on more informative words but this strategy will struggle with out-of-vocabulary words.

2 Another way to make the data more linearly separable is with word embeddings which use pre-trained word embeddings to represent words as dense vectors. This captures semantic relationships between words (i.e. "good" VS "not good"). This improves generalization by using semantic knowledge but it can introduce bias present in the pre-training corpus. □

Problem 2: 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.

Solution. **1** Probability y=1 for logisitic regression is:
$P(y = 1|x; w) = \sigma(w^T\phi(x))$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

2 Likelihood function is:
$L(w) = \Pi_{i=1}^n P(y^{(i)}|x^{(i)}; w)$

3 Log-likelihood is:
$\ell(w) = \log(L(w)) = \sum_{i=1}^n \log(P(y^{(i)}|x^{(i)}; w))$

4 Expand:
$\ell(w) = \sum_{i=1}^n [y^{(i)} \log \sigma(w^T\phi(x^{(i)})) + (1 + y^{(i)}) \log(1 - \sigma(w^T\phi(x^{(i)})))]$

5 Take minimum:
$\min_w - \sum_{i=1}^n [y^{(i)} \log \sigma(w^T\phi(x^{(i)})) + (1 + y^{(i)}) \log(1 - \sigma(w^T\phi(x^{(i)})))]$ □

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.

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.

Solution. **1** From the solution to Problem 2.1:
$\ell_i(w) = -[y^{(i)} \log f_w(x^{(i)}) + (1 - y^{(i)}) \log(1 - f_w(x^{(i)}))]$
where $f_w(x) = \frac{1}{1+e^{-w\cdot\phi(x)}}$

2 Differentiate $\ell_i(w)$ with respect to w to compute the gradient:
$\nabla_w \ell_i(w) = -[y^{(i)} \nabla_w \log f_w(x^{(i)}) + (1 - y^{(i)}) \nabla_w \log(1 - f_w(x^{(i)}))]$

3 Compute gradients (part 1):
$\nabla_w \log f_w(x^{(i)}) : \nabla_w \log f_w(x^{(i)}) = \frac{1}{f_w(x^{(i)})} \cdot \nabla_w f_w(x^{(i)}) \nabla_w f_w(x^{(i)})$
$= f_w(x^{(i)}) \cdot (1 - f_w(x^{(i)})) \cdot \phi(x^{(i)})$
$\Rightarrow \nabla_w \log f_w(x^{(i)})$
$= (1 - f_w(x^{(i)})) \cdot \phi(x^{(i)})$

4 Compute gradients (part 2):
$\nabla_w \log(1 - f_w(x^{(i)})) : \nabla_w \log(1 - f_w(x^{(i)}))$
$= \frac{-1}{1-f_w(x^{(i)})} \cdot \nabla_w f_w(x^{(i)})$
$= \frac{-1}{1-f_w(x^{(i)})} \cdot f_w(x^{(i)}) \cdot (1 - f_w(x^{(i)})) \cdot \phi(x^{(i)})$
$= -f_w(x^{(i)}) \cdot \phi(x^{(i)})$

5 Substitute:
$\nabla_w \ell_i(w) = -[y^{(i)}(1 - f_w(x^{(i)})) \cdot \phi(x^{(i)}) + (1 - y^{(i)})(-f_w(x^{(i)}) \cdot \phi(x^{(i)})]$
$= -[(y^{(i)} - y^{(i)} f_w(x^{(i)}) - f_w(x^{(i)}) + y^{(i)} f_w(x^{(i)}) \cdot \phi(x^{(i)})]$
$= -[y^{(i)} - f_w(x^{(i)}) \cdot \phi(x^{(i)})]$

6 Therefore:
$\nabla_w \ell_i(w) = (f_w(x^{(i)}) - y^{(i)}) \cdot \phi(x^{(i)})$ □

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.

Solution. **1** Some potential problems are:
- Unigram loses semantic meaning because the independent unigram tokens ignore dependent word combinations like the ”good” VS ”not good” example, and also because it ignores word order which is important for context and meaning.
- Unigram gives equal importance to all tokens, so words like ”the” and ”a” are weighted the same as words that provide high context.
- There is a potential for overfitting when working with a large dataset.
- Unigram cannot handle out-of-vocabulary words.

2 Some fixes to address these issues are:
- Including bigrams to account for word order information and dependent word combinations.
- Using TF-IDF to give more weight to important words.
- Define specific key words to account for negation. □

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.

Solution. **1** P: An older gentleman speaking at a podium .
H: A man giving a speech
This example is missing the context of "speaking at a podiun" = "speech". To fix this, the system either needs more ngrams to gather more extensive context beyond two tokens and/or it needs pre-training on a corpus where it can learn that "speaking"+"podium"="speech".

2 P: The woman in the blue skirt is sleeping on a cardboard box under a picture of Mary and baby Jesus .
H: A person sleeping on a cardboard box below a picture of religious icons .
The weights pertaining to "Mary", "baby Jesus", "icons", and "religious icons" all equal zero which indicates that the system doesn't understand who Mary and baby Jesus are. To fix this, the system needs training on a corpus that can provide the context that Mary and baby Jesus are religious icons.

3 P: A couple bows their head as a man in a decorative robe reads from a scroll in Asia with a black late model station wagon in the background .
H: A black late model station wagon is in the background .
High weights are given to the words not in the hypothesis (details about the couple, robe, Asia, and scroll) and low weights were given to bigrams about the car. This issue is likely caused by the use of TF-IDF which gives higher weight to unique words which would be the words not repeated in the hypothesis. We would need to alter the system to look for matching words instead to solve the issue with this example.

4 P: Two guys are wearing uniforms who are running in the grass.
H: Two guys in uniforms are playing a sport in the grass.
"running" and "sport" have high but opposite signed weights so they negate each other. This example needs more context from pre-training that sports involve running.

5 P: A woman in a purple dress talks on her cellphone and a man reads a book as a two-story bus passes by their window .
H: A woman calls her kids
This example is missing the context between "talk"+"cellphone"="call" and there is no mention of children in the premise. This requires pre-training for the added context that "talk"+"cellphone"="call" and there also needs to be an out-of-vocab handling method that gives more weight to words in or semantically similar to the ones in the vocab and lower weights to words not at all in the vocab. ☐

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.

Solution. **1** When only using the hypothesis, the accuracy actually increases.

2 While this is not what I expected, I would guess that having a simpler, smaller vocabulary helps with the issue of overfitting that occurs when also using the premise which tends to be text heavy and often contains unnecessary vocabulary. □

Problem 3: 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.

Solution. No, we cannot set hermitian=True. Despite the co-occurrence matrix being symmetrical, it is not necessarily equal to its own conjugate transpose which is required for it to be considered hermitian. □

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.

Solution. **1** Words other gendered terms and names - words such as ”one”, ”be”, etc. and also punctuation - are present in the top-k words.

2 This is likely because the dot product is biased towards high magnitude vectors. So, the most frequent words are more likely to be in the top-k rather than the most similarly semantic top-k words. □

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.

Solution. **1** There are a few tokens that differ between ”man” and ”woman”. Only ”man” contains: woman, moment, and farmer. And, only ”woman” contains: man, people, and circumstance.

2 Words like ”gentleman” and ”farmer” make sense to be in the top-k words for ”man” as those are terms/occupations associated with men typically. And, words like ”lady” and ”girl” make sense to be in the top-k words for ”woman” as those are typically terms associated with women.

3 What doesn’t make sense is the opposite gendered words appearing among the top-k words, like ”lady” for ”man” and ”gentleman” for ”woman”. My guess would be that these terms are considered to be semantically similar since they refer to a gendered term. □

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?

Solution. **1** The top-k words are mostly synonyms with one or two antonyms.

2 I would expect to see primarily synonyms especially if ”happy” and ”sad” are used in pairs in the text like ”happy and agreeable” or ”sad and painful”. A smaller number of antonyms is also expected in the case that they are used as a comparison in the text like ”happy, rather than distressed” or ”sad, rather than delightful”. □