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

Representing and Classifying Text

Name
NYU ID

Please write down any collaborators, AI tools (ChatGPT, Copliot, codex, etc.), and external resources you used for this assignment here.
Collaborators:
AI tools:
Resources:

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: 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.

The decision boundary can be derived as the following:

$$p_{\vec{\theta}}(y=1|\vec{x}) - p_{\vec{\theta}}(y=0|\vec{x})$$

By bayes rule we have: $\frac{p_{\vec{\theta}}(\vec{x}|y=1)p_{\vec{\theta}}(y=1)}{p_{\vec{\theta}}(\vec{x})} = \frac{p_{\vec{\theta}}(\vec{x}|y=0)p_{\vec{\theta}}(y=0)}{p_{\vec{\theta}}(\vec{x})}$ And by assumption $p_{\vec{\theta}}(y=0) = p_{\vec{\theta}}(y=1)$ we can derive the following:

$$f(\vec{x}) = p_{\vec{\theta}}(\vec{x}|y=1) - p_{\vec{\theta}}(\vec{x}|y=0)$$

2. [3 points] Recall that for multinomial Naive Bayes, we have the input $X = (X_1, \dots, X_n)$ 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 $X_i \mid Y = y \sim \text{Categorical}(\theta_{w_1,y}, \dots \theta_{w_m,y})$ 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

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

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

Solution: Assume that $P(y=1) = \theta_1 \quad P(y-0) = 1-\theta_1$ We only need to solve the expression:

$$\begin{aligned}
p_\theta(\vec{x} \mid y=1)p(y=1) &= p_{th\vec{e}ta}(\vec{x} \mid y=0)p(y=0) \\
\prod_{i=1}^{n} p_{\vec{\theta}}\left(\vec{x}_i \mid y=1\right)\theta_1 &= \prod_{i=1}^{n} p_{\vec{\theta}}\left(\vec{x}_i \mid y=0\right)(1-\theta_1) \\
\left(\prod_{i=1}^{n}\prod_{j=1}^{m} \theta_{w_{j,1}} x_{i,j}\right)\theta_1 &= \left(\prod_{i=1}^{n}\prod_{j=1}^{m} \theta_{w_{j,0}} x_{i,j}\right)(1-\theta_1) \\
\sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j}\log\left(\theta_{w_{j,1}}\right) + \log\theta_1 &= \sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j}\log\left(\theta_{w_{j,0}}\right) + \log(1-\theta_1)
\end{aligned}$$

where $x_{i,j} = \begin{cases} 1 & x_{i,j} = w_i \\ 0 & otherwise \end{cases}$, $\sum_{j=1}^m \theta_{w_{j,1}} = 1, \sum_{j=1}^m \theta_{w_{j,0}} = 1$ We have

$$\begin{aligned}
h(\vec{x}) &= left - right \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j}\log\frac{\theta w_{j,1}}{\theta w_{j,0}} + \log\frac{\theta_1}{1-\theta_1} \\
&= \vec{\omega}^\top \vec{x} + b
\end{aligned}$$

where $\vec{\omega} = \left\{(\omega_{1,1}, \cdots \omega_{n,m}) \,\middle|\, \omega_{i,j} = \log\frac{\theta_{\omega_{j,1}}}{\theta_{\omega_{j,0}}}\right\}$ and $b = log\frac{\theta_1}{1-\theta_1}$.

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?

   The assumption behind Naive Bayes Classifier is that all the features (words in this case) are conditionally independent given the class label $y \in \{0, 1\}$. However, this assumption might not hold true in many real-world text classification tasks, since words in a sentence or document are often dependent on one another. For example, when we are doing sentiment analysis, we may encounter such bi-gram "not satisfactory" where "not" is a negative word (which only has meaning when coupled with other adjective) and "satisfactory" is a positive word. But with independence assumption, we may fail to capture "not satisfactory" as a whole to be negative and instead only capture "satisfactory" as a positive word and predict the sentence as positive.
   In order to alleviate the problem, i could come up with two ways:

   (a) Use n-gram model to capture meaning of a group of words rather than single ones.

   (b) Use word embeddings which is learnt from large amount of corpus. Similar words are close to each other in the continuous vector space that is learnt and that word vectors can capture semantic meaning of a word in a document.

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. Several ways could be applied to make datasets more linearly separable:

   (a) **Use word embeddings** to more accurately represent a sentence so that similar sentences (semantically) are close to each other in the vector space. For example, if we represent each sentence by averaging the word vectors of each of its composing words, we know that similar sentences would be clustered in the vector space, making it easier for datasets to be linear separable.

   (b) **Delete the stop words** from each sentence. This could reduce the noise of the dataset. Without these frequent, non-informative words, the model can thus focus on the meaningful content of each sentence, resulting in a more linear separable datasets.

   (c) **Feature Engineering**: We could use kernel method to create more features for each sentence other than embedding itself. For example we could encode the length, duplicacy information within the sentence into the feature vector. Sometimes even if the dataset is not linearly separable in lower dimension, it will be in higher dimensional space. Similarly, we could **apply SVD or PCA** to the dataset for similar reasoning.

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.

We have the following derivations:

$$\begin{aligned}
h\left(\vec{x}^{(i)}\right) &= \vec{\omega}^T \cdot \phi\left(\vec{x}^{(i)}\right) + b && (1)\\
z\left(\vec{x}^{(i)}\right) &= \frac{1}{1 + e^{-h\left(\vec{x}^{(i)}\right)}} && (2)\\
l(\vec{\omega}) &= \prod_{i=1}^{n} z\left(\vec{x}^{(i)}\right)^{y^{(i)}} \left(1 - z\left(\vec{x}^{(i)}\right)\right)^{1-y^{(i)}} && (3)\\
l(\vec{\omega}) &= \sum_{i=1}^{n} y^{(i)} \log z\left(\vec{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - z\left(\vec{x}^{(i)}\right)\right) && (4)\\
\max_{\vec{\omega}} l(\vec{\omega}) &= \min_{\vec{\omega}} -l(\vec{\omega}) && (5)\\
&= \min_{\vec{\omega}} -y^{(i)} \log\left(\frac{1}{1 + e^{-z\left(\vec{x}^{(i)}\right)}}\right) - \left(1 - y^{(i)}\right) \log\left(\frac{e^{-z\left(\vec{x}^{(i)}\right)}}{1 + e^{-z\left(\vec{x}^{(i)}\right)}}\right) && (6)
\end{aligned}$$

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.

See the corresponding part in submission.py

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.

$$l_i(\omega) = -y^{(i)} \log \frac{1}{1+e^{-\omega\cdot\phi\left(x^{(i)}\right)}} - \left(1-y^{(i)}\right) \log\left(1 - \frac{1}{1+e^{-\omega\cdot\phi\left(x^{(i)}\right)}}\right) \tag{7}$$

$$\text{Denote } \sigma(z) = \frac{1}{1+e^{-z}} \tag{8}$$

$$\nabla_z \log\sigma(z) = 1-\sigma(z) \tag{9}$$

$$\nabla_z \log(1-\sigma(z)) = -\sigma(z) \tag{10}$$

$$\therefore \nabla_w l_i(\omega) = y^{(i)}\phi\left(x^{(i)}\right) \cdot \frac{e^{-\omega\cdot\phi\left(x^{(i)}\right)}}{1+e^{-\omega\cdot\phi\left(x^{(i)}\right)}} \tag{11}$$

$$+ \left(1-y^{(i)}\right)\phi\left(x^{(i)}\right) \cdot \frac{1}{1+e^{-\omega\cdot\phi\left(x^{(i)}\right)}} \tag{12}$$

$$= \left(\sigma\left(\omega\cdot\phi\left(x^{(i)}\right)\right) - y^{(i)}\right)\phi\left(x^{(i)}\right) \tag{13}$$

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*.

See the corresponding part in submission.py

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

(a) **Potential Problem:**
   i. Unigram features include a lot of common words (like "the," "is," "of"), which often don't contribute significantly to distinguishing between classes.
   ii. Unigram captures each word independently without considering the context provided by neighboring words. This leads to a loss of semantic information that is often crucial for understanding nuances in sentences.

(b) **Better Design:**
   i. Using N-gram feature extractor is a step toward addressing the limitations of the unigram model, providing a local context around each word
   ii. Incorporating tf-idf into the feature encoding can balance word frequency and reduce noise from common words.

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.

I choose Ngram model for better generalization performance. See the corresponding part in submission.py

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.

(a) **Example 1**: "An older gentleman speaking at a podium." vs. "A man giving a speech." where label is 0 and predicted is 1.

  i. **Intuition:** The classifier likely focused on the similarity between "giving a speech" and "speaking at a podium," leading it to mistakenly predict an entailment. It ignored the critical context of "older gentleman" versus just "man," as well as the subtle difference in meaning between simply speaking and giving a speech.

  ii. **Additional Information Needed:** A deeper semantic understanding or more fine-grained context capturing the differences between "older gentleman" and "man" could help. Injecting phrase-level semantics instead of 2-gram would be necessary to catch such distinctions.

(b) **Example 2**:"A busy street full of shops and people holding hands and walking." vs. "Couples looking for places to shop."

  i. **Intuition:** The 2-gram model lacks the ability to distinguish the actions ("holding hands" and "walking") from actively "looking for places to shop."

  ii. **Additional Information Needed**: Better handling of context-specific actions and activities. A model with an improved representation of relationships between words and actions would likely help.

(c) **Example 3:** "The woman in the blue skirt is sleeping on a cardboard box under a picture of Mary and baby Jesus." vs. "A person sleeping on a cardboard box below a picture of religious icons."

  i. **Intuition:** Interesting one. Even if the model correctly classified this , I don't think the model is understanding this classification since many of the 2-grams had zero weights (e.g., "picture Mary," "Mary baby," "religious icons"). This indicates that while the model recognized some semantics, it failed to leverage other key semantics(e.g. religious references).

  ii. **Additional Information Needed:** More nuanced handling of religious terminology and synonyms for "icons" (e.g., "Mary," "baby Jesus"). External knowledge sources or semantic embeddings are required for better learnability.

(d) **Example 4:** "A cabin shot of a very crowded airplane." vs. "The airplane is quite crowded."

  i. **Intuition:** The classifier may not have recognized that "cabin shot" and "airplane" are closely related. The 2-gram model seems to have missed the semantic equivalence between "very crowded airplane" and "airplane is quite crowded," possibly due to limited overlap in word pairs.

  ii. **Additional Information Needed:** Improved word embeddings that capture synonymy (e.g., "very crowded" and "quite crowded") and the understanding that "cabin" implies an airplane context. Enhanced contextual modeling would be beneficial.

(e) **Example 5:** "A boy plays in the surf." vs. "The boy is wearing red trunks."

  i. **Intuition:** The 2-gram model might have fixated on the phrase "The boy," failing to recognize the contradiction between the boy playing in the surf and wearing red trunks. It likely associated words related to "boy" and missed the specific contexts differentiating the two statements.

  ii. **Additional Information Needed:** Contextual phrase analysis or better entity recognition (i.e., understanding that "playing in the surf" has no direct indication of wearing "red trunks"). Including a mechanism for tracking action-related words (e.g., "playing") in conjunction with descriptions would help.

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.

Intuitively, since we encode less information in our features, the accuracy is expected to go down. By experiment, the train_err and validation_err both goes up and the accuracy both goes down. This is expected since we are not utilize all the information for each data point and the feature that we extract are not enough to capture the relationship between hypothesis, premise and label.

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).

See the corresponding function in submission.py

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 we can, since the co-occurrence matrix is always read-valued and symmetric so by the documentation we can perform SVD more efficiently.

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

See the corresponding function in submission.py

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'`).

See the corresponding code in submission.py

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

We observe that there are many punctuation in the top-k similar words, which doesn't seem similar at all. For example, the top-10 similar words predicted for "man" is given by [and i it the " in she but that]. This is because without normalization, some word embedding vectors have larger magnitude than others even with the same direction. This could cause some irrelevant words to be more similar to the target word due to its larger magnitude.

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

See the corresponding code in submission.py.

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.

From the embedding model we get the following results:

(a) **The top-10 similar words for ”man”**: [woman lady farmer charming _woman_ pert gallant ladies weak fine]

(b) **Top k most similar to ”women”**:[man charming farmer pert lady girl blush fine minute sweet]

The similar words between ”man” and ”woman” generally make sense, as ”charming,” ”pert,” ”fine,” and shared gender-related terms reflect contexts where these words co-occur. Gendered terms like ”lady” and ”girl” make sense as close terms to ”woman.” However, the appearance of words like ”weak” in the context of ”man” and ”minute” for ”woman” raises questions about the data that informed these embeddings, suggesting that embeddings can sometimes reflect unexpected or biased relationships based on the training corpus. These word embeddings might reflect stereotypical biases present in the data they were trained on.

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?

From the embedding model we get the following results:

(a) **The top-10 similar words for ”happy”**: [distressing agreeable superior sorry likely easy unpleasant interesting inferior pleasant]

(b) **Top k most similar to ”sad”**:[little different few quivering lame momentary large sudden small delightful]

We could observe that for ”happy”, we observe a mix of synonyms (e.g., ”pleasant”, ”agreeable”, ”easy”) and antonyms (e.g., ”distressing”, ”unpleasant”, ”sorry”, ”inferior”). For ”sad”, the words seem more neutral(like ”little”, ”few”, ”different”) or unrelated to the concept of sadness.

The presence of both synonyms and antonyms in the similar words for ”happy” is surprising. Ideally, we’d expect mostly synonyms for similar word embeddings, but antonyms being ranked highly suggests that the embeddings capture contextual co-occurrences rather than pure semantic similarity.

Personally I think this is due to that fact that word embedding is trained by context window where words that are contextually(or positionally) close to each each other are considered similar to each other. This contextual similarity could explain why we see both synonyms and antonyms in the top-10 similar word for the word ”happy”.