DS-GA-1011: Natural Language Processing with Representation Learning, Fall 2024 Representing and Classifying Text

Yusuf Baig
N17203193

Please write down any collaborators, AI tools (ChatGPT, Copilot, codex, etc.), and external resources you used for this assignment here.
Collaborators:
AI tools: ChatGPT
Resouces: cs.cmu.edu, web.stanford.edu

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 specified 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.

Ans: In binary classification with classes $y \in \{0,1\}$, Naïve bayes uses the Bayes theorem to model the conditional probability.

Using Bayes Theorem:

$$p_\theta(y|x) = p_\theta(x|y)\frac{p_\theta(y)}{p_\theta(x)}$$

Since $h(x)$ is the decision boundary function, it should occur when both the classes are equally likely:

$$p_\theta(y=1|x) = p_\theta(y=0|x)$$

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

For the multinomial Naive Bayes model, where the features are conditionally independent given the class label, the likelihood $p_\theta(x|y)$ can be written as a product of the conditional probabilities for each feature x_i:

$$p_\theta(x \mid y) = \prod_{i=1}^{n} p_\theta(x_i|y)$$

To avoid underflow since there are (or could be) many features, we take the logarithm of the probabilities, turning it into a sum making it easier to compute and compare.

$$\log\left(\frac{p_\theta(y=1|x)}{p_\theta(y=0|x)}\right) = \log\left(\frac{p_\theta(x|y=1)p_\theta(y=1)}{p_\theta(x|y=0)p_\theta(y=0)}\right)$$

$$\log\left(\frac{p_\theta(y=1)}{p_\theta(y=0)}\right) + \log\left(\frac{p_\theta(x|y=1)}{p_\theta(x|y=0)}\right) = 0$$

Thus, the equation for *h(x)* becomes:

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

2. [3 points] Recall that for multinomial Naive Bayes, we have the input $X = (X_1,...,X_n)$ 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

$X_i \mid Y = y \sim$ Categorical($\theta_{w1,y},\dots\theta_{wm,y}$) where $Y \in \{0,1\}$, $w_i \in$ V, and $m = |V|$ is the vocabulary size. Further, $Y \sim$ Bernoulli(θ_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

3.

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

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

Ans: Using the equation that we got above we can use the equation of *h(x)* which is defined as :

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

Considering our assumption for Naïve Bayes holds true, the likelihood *P(X|Y=y)* is the product of individual word probabilities, assuming the word counts of X_i is conditionally independent given *Y*.

$$p(X = x|Y = y) = \prod_{i=1}^{m} \theta_{x^i,y}$$

Hence if we take the logs likelihood ratio:

$$\log\left(\frac{p_\theta(X=x|Y=1)}{p_\theta(X=x|Y=0)}\right) = \sum_{i=1}^{n} \log\left(\theta_{x^i,1}/\theta_{x^i,0}\right)$$

And since Y follows a Bernoulli(θ_1) distribution we can include the prior probability for Y, where θ_1 is the probability of Y=1.

$$\log\left(\frac{p_\theta(Y=1)}{p_\theta(Y=0)}\right) = \log\left(\frac{\theta_1}{1-\theta_1}\right)$$

Therefore, the final equation of $h(x)$ combining the log-odds equation and prior probability comes out to be:

$$h(x) = \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{i=1}^{n} \log\left(\theta_{x^i,1}/\theta_{x^i,0}\right)$$

We can show that the decision boundary can be written in the form of $w.x + b = 0$, therefore we can express this as:

$$h(x) = w.x + b$$

Where:

w = (log $\left(\frac{\theta_{1,1}}{\theta_{1,0}}\right)$, …., log $(\theta_{m,1}/\theta_{m,0})$)

and $b = \log\left(\frac{\theta_1}{1-\theta_1}\right)$

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 alleviate the problem?

Ans: A common example we can use for this question is Sentiment Analysis of Movie Reviews. The Naïve Bayes treats each word of the sentence independently since it's a unigram model, which can lead to different meanings of certain word combinations changing the meaning of the phrases/sentences significantly. For Example:

- A sentence like "not a good movie at all" expresses a negative sentiment, but Naïve Bayes would treat "not" and "good" independently. Since the word "good" is associated with a positive sentiment, the classifier could misclassify it.

To address this problem, we can use a few methods like:

- Using n-grams instead of unigrams: Instead of treating it as independent, we can use bi or trigrams as features. This will allow the model to capture dependencies between words, a word pair like "not good" will be treated as a single feature, helping the model classify it correctly.
- Feature Engineering: Manually crafting features that take into account common word dependencies like negation handling, combing "not" with the consequent words could improve classification without dramatically increasing the feature space.

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.

Ans: To make the data more linearly separable we can try out the following ways:

- Feature Engineering:
    - TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF down-weights common words (like stop words) and highlights important words, making the data more separable in the feature space.
    - Word Embeddings: Embeddings help create a more structured feature space, where semantically similar words are close together, enhancing linear separability.
- Dimension Reduction:
    - Principal Component Analysis (PCA) or Singular Value Decomposition (SVD) can be used to reduce the dimensionality of text data while retaining important information, focusing on the most important features and removing noisy or irrelevant ones, dimensionality reduction can make the data more linearly separable.
- Text Preprocessing:
    - Removing Stopwords: Reducing noise in the data by removing irrelevant words helps the classifier focus on meaningful words, enhancing linear separability.
    - Stemming and Lemmatization: Balancing the class distribution makes the decision boundary more stable, improving linear separability.

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.

Ans: The logistic regression model is used to predict a binary outcome. The Label $y^{(i)}$ is either 1 or 0.
We know that the logistic regression function uses a sigmoid function to map the feature vector $\phi(x^{(i)}).w$ to a probability of the label being 1(entailment). The function is defined as:

$$\sigma(z) = 1/(1 + e^{-z})$$

For each example *i*, the model predicts a probability of entailment $\hat{y}^{(i)}$ is given by:

$$\hat{y}^{(i)} = \sigma(\ \phi(x^{(i)}).w = 1/1 + e^{-\phi(x^{(i)}).w}$$

The log-likelihood for a single data point *i* is the probability assigned to the correct label. For binary classification, the log-likelihood for a data point is:

$$logP(y^{(i)}|x^{(i)}) = \ y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})$$

The maximum likelihood function objective for their entire dataset is the sum of the log-likelihoods across all data points:

$$L(w) = \sum_{i=1}^{n} [\, y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})$$

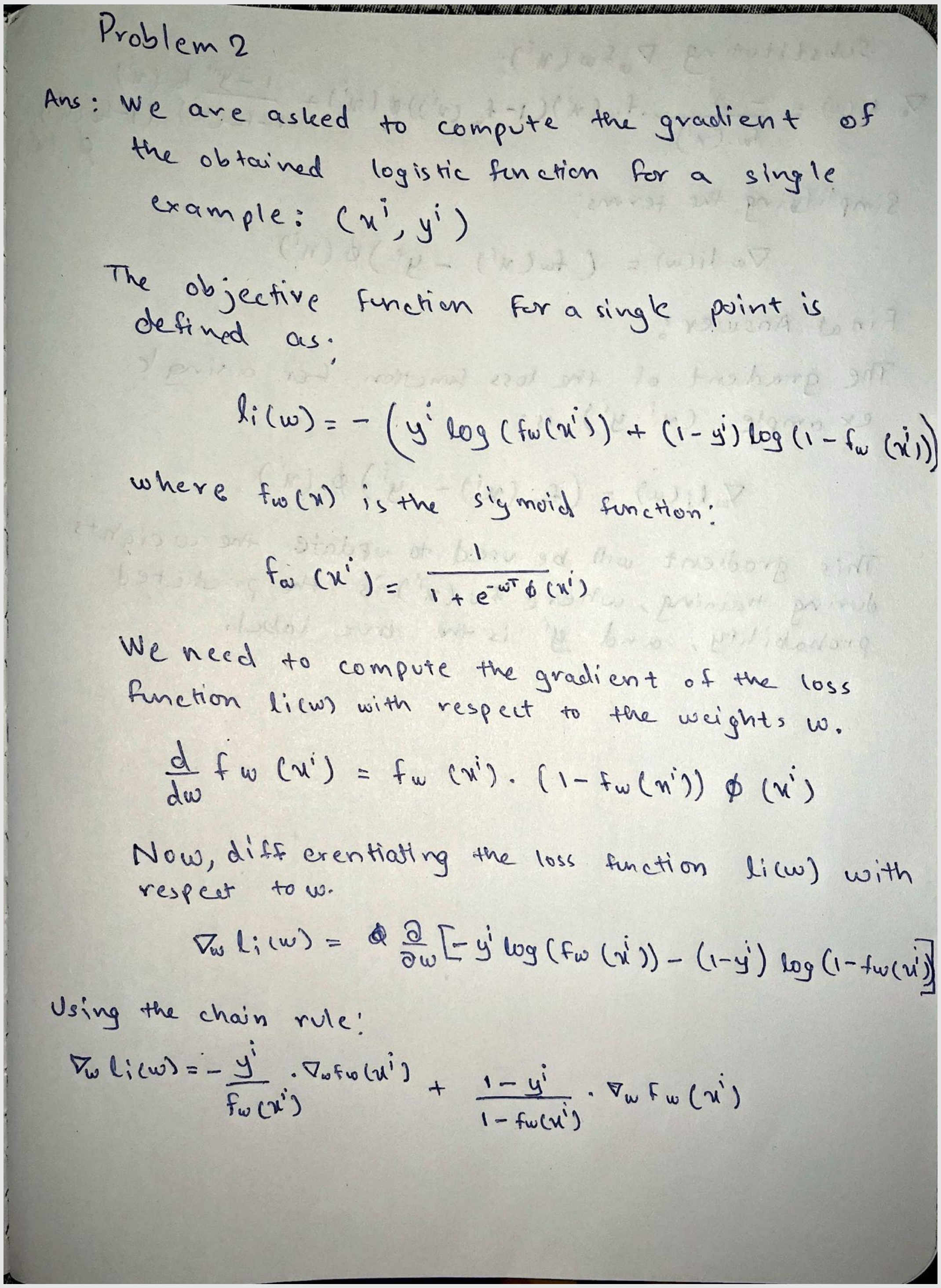
This is the function we want to maximize with respect to *w.* To turn this into a minimization problem, we simply minimize the negative log-likelihood:

$$L(w) = -\sum_{i=1}^{n} [\, y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})$$

This is the objective function to minimize in logistic regression.
Where $\hat{y}^{(i)} = \sigma(\ \phi(x^{(i)}).w$

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.

Ans: 

Problem 2

Ans: We are asked to compute the gradient of the obtained logistic function for a single example: (x^i, y^i)

The objective function for a single point is defined as:

$$l_i(w) = -\left(y^i \log(f_w(x^i)) + (1-y^i)\log(1-f_w(x^i))\right)$$

where $f_w(x)$ is the sigmoid function:

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

We need to compute the gradient of the loss function $l_i(w)$ with respect to the weights w.

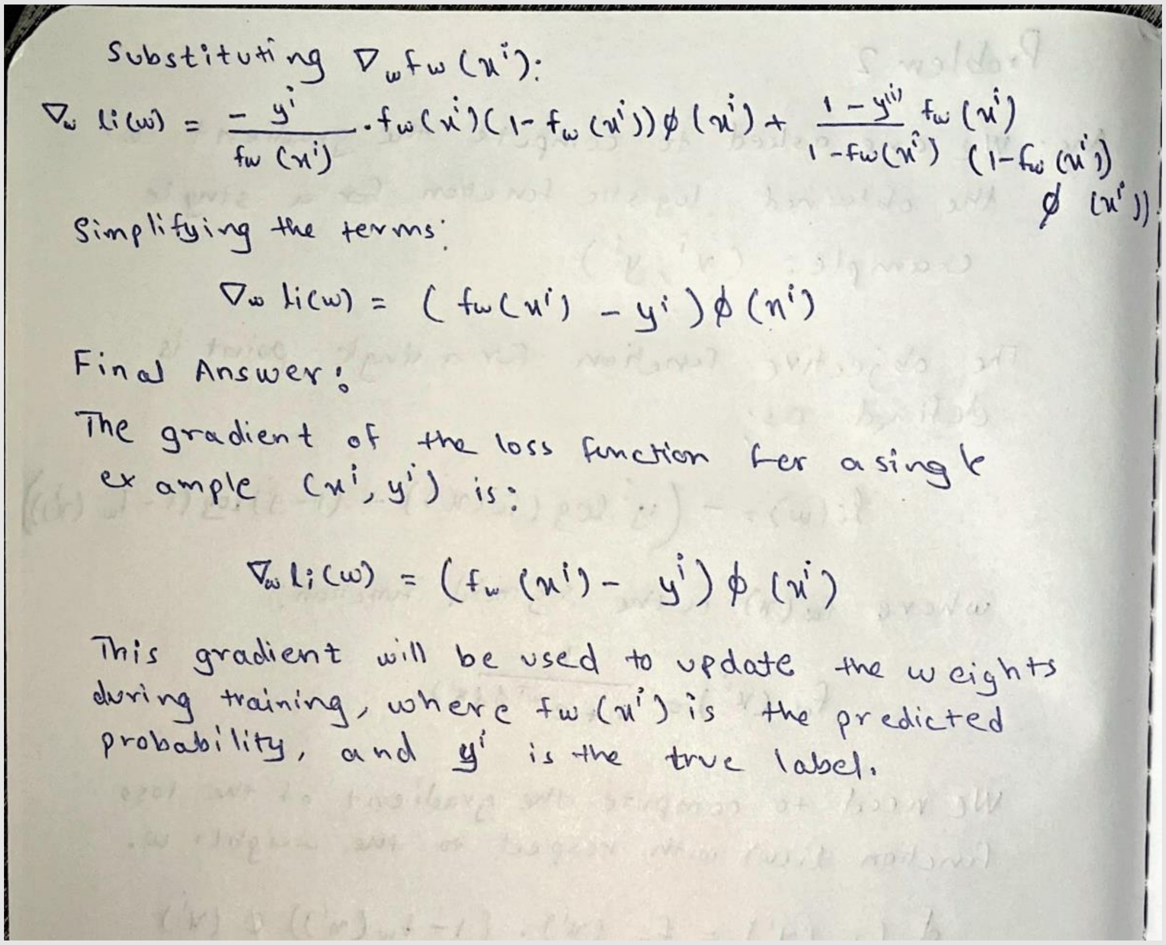
$$\frac{d}{dw} f_w(x^i) = f_w(x^i)\cdot(1-f_w(x^i))\,\phi(x^i)$$

Now, differentiating the loss function $l_i(w)$ with respect to w.

$$\nabla_w l_i(w) = \cancel{\phi}\,\frac{\partial}{\partial w}\left[-y^i\log(f_w(x^i)) - (1-y^i)\log(1-f_w(x^i))\right]$$

Using the chain rule:

$$\nabla_w l_i(w) = -\frac{y^i}{f_w(x^i)}\cdot\nabla_w f_w(x^i) + \frac{1-y^i}{1-f_w(x^i)}\cdot\nabla_w f_w(x^i)$$



Substituting $\nabla_w f_w(x^i)$:

$$\nabla_w l_i(w) = \frac{-y^i}{f_w(x^i)} \cdot f_w(x^i)(1-f_w(x^i))\phi(x^i) + \frac{1-y^i}{1-f_w(x^i)} f_w(x^i)(1-f_w(x^i))\phi(x^i))$$

Simplifying the terms:

$$\nabla_w l_i(w) = (f_w(x^i) - y^i)\phi(x^i)$$

Final Answer:

The gradient of the loss function for a single example (x^i, y^i) is:

$$\nabla_w l_i(w) = (f_w(x^i) - y^i)\phi(x^i)$$

This gradient will be used to update the weights during training, where $f_w(x^i)$ is the predicted probability, and $\underline{y^i}$ is the true label.

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.

Ans:

- **Loss of Word Order Information**:
  - Unigram feature extractors represent each sentence as a Bag-of-Words (BoW), ignoring the order of the words. This can lead to a loss of crucial syntactic and semantic information. Example: The sentences "The dog bit the man" and "The man bit the dog" would have identical unigram features, but their meanings are very different.
- **Failure to Handle Polysemy and Synonyms**:
  - Words with multiple meanings (polysemy) or synonymous words are treated as distinct in the unigram model. This can result in poor generalization. For instance, "big" and "large" would generate different features, even though they have similar meanings in many contexts.
- **Sparse Feature Space**:
  - The feature space grows quickly as more unigrams are added, leading to a sparse and high-dimensional feature vector, which can make training less efficient and prone to overfitting, especially when there is limited data.
- **No Capture of Semantic Similarity**:
  - Unigram models treat words as independent tokens. Words that are semantically related but have different surface forms (e.g., "run" and "ran") are considered completely different in this model.

To address the limitations of unigram features, a more sophisticated feature extractor can be designed to capture richer semantic and syntactic information. Here are potential enhancements:

- **Bigram and N-gram Features:**
  - Bigrams (and other n-grams) capture adjacent word pairs or sequences, preserving some word order and contextual information that unigrams miss. Include not only unigrams but also bigrams (pairs of consecutive words) and potentially higher-order n-grams (e.g., trigrams) to capture word dependencies.
- **TF-IDF (Term Frequency-Inverse Document Frequency):**
- Raw word counts can lead to over-representation of common words like "the," "is," etc. Using TF-IDF weighs words by their importance, down weighting common words and emphasizing rare but meaningful words. Instead of just counting unigrams and n-grams, compute the TF-IDF score for each word or n-gram, reflecting how important a word is in a given sentence relative to the entire dataset.
- **Word Embeddings (e.g., Word2Vec, GloVe, FastText):**
- Word embeddings map words to dense, low-dimensional vectors, where semantically similar words are placed closer together in vector space. This addresses the issue of synonyms and polysemy by encoding semantic meaning. Instead of one-hot encoding or counting words, use pre-trained embeddings to represent each word as a vector. Aggregate the embeddings of all words in a sentence (e.g., by averaging) to create a sentence-level feature vector.
- **Syntactic Parsing and Dependency Features:**
- Relationships between words (such as subject-verb-object) are important for determining entailment, but these are missed by unigram models. Use syntactic parsers to extract features based on grammatical structure and dependencies between words. For instance, dependency parsing can reveal that "the man" is the subject and "bit" is the action in "The man bit the dog."
- **Interaction Features (Cross-features):**
- Capturing relationships between the premise and hypothesis is crucial for textual entailment. Generate cross-features by computing interactions between premise and hypothesis words. For instance, you could compute word overlaps or cosine similarity between the premise and hypothesis embeddings. These interactions help capture direct entailment relationships.

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.

Ans:

Example 1:

- Premise: "An older gentleman speaking at a podium."
- Hypothesis: "A man giving a speech."
- Label: 0, Predicted: 1, Wrong
- The classifier likely over-relied on shared terms like "man" and "speech" in the hypothesis and "gentleman" and "speaking" in the premise. It did not capture the key difference between "speaking" and "giving a speech" or the nuanced change in meaning between "older gentleman" and "man." The model needs to differentiate actions better and consider more specific entity descriptions like age. To fix this, adding semantic role labeling and focusing on adjective modifiers would help the model capture these subtle distinctions.

Example 2:

- Premise: "The woman in the blue skirt is sleeping on a cardboard box under a picture of Mary and baby Jesus."
- Hypothesis: "A person sleeping on a cardboard box below a picture of religious icons."
- Label: 1, Predicted: 0, Wrong
- The model failed to recognize that "Mary and baby Jesus" and "religious icons" refer to the same concept. Although both sentences describe a person sleeping on a cardboard box under a picture, the model struggled to connect the religious figures with their generalized category. This error highlights a lack of semantic similarity recognition between proper nouns and their more general categories. The model could benefit from using entity resolution to map specific names to broader categories. Additionally, synonym detection between words like "Mary and baby Jesus" and "religious icons" would improve predictions.

Example 3:

- Premise: "A cabin shot of a very crowded airplane."
- Hypothesis: "The airplane is quite crowded."
- Label: 1, Predicted: 0 Wrong
- The classifier likely missed the similarity between "cabin shot of a crowded airplane" and "the airplane is quite crowded." It could not bridge the connection between the visual description and the general statement. The model needs better handling of paraphrases, especially when it comes to descriptions that focus on visual elements and their simplified counterparts. Sentence-level paraphrase detection and more sophisticated handling of context could help bridge this gap.

Example 4:

- Premise: "A building that portrays beautiful architecture stands in the sunlight as somebody on a bike passes by."
- Hypothesis: "A bicyclist passes an esthetically beautiful building on a sunny day."
- Label: 1, Predicted: 0, Wrong
- The classifier missed the paraphrasing between "stands in the sunlight" and "on a sunny day" and between "somebody on a bike" and "bicyclist." The model failed to connect "beautiful architecture" with "a beautiful building," likely because it doesn't fully recognize these as equivalent expressions. The model needs to be better at paraphrase recognition, especially for abstract concepts like beauty and weather descriptions. The model also needs to better understand synonyms for actions and descriptions, Incorporating contextual embeddings (like BERT or GPT), or just stemming and lemmetization would help the model recognize these paraphrases.

Example 5:

- Premise: "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."
- Hypothesis: "A black late model station wagon is in the background."
- Label: 1, Predicted: 0, Wrong
- The classifier likely focused too much on the less important parts of the premise, such as the couple and the man reading from a scroll, rather than on the black station wagon in the background, which is the key shared element between the premise and the hypothesis. The model needs better focus on key elements within the sentence. It needs to give more weight to the entities that are mentioned in both the premise and hypothesis. Incorporating entity importance ranking or improving its ability to focus on specific objects described in the premise and hypothesis could 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.

Ans: The error rate for the dev set decreased, something that was not expected, this could be happening due to the following reasons:

- Overfitting
  - Overfitting to the irrelevant features from the premise, when using both the premise and hypothesis for feature extraction, the model might overfit to irrelevant features in the premise that don't contribute to the entailment or contradiction classification. By removing the premise and only focusing on the hypothesis, the model could be learning more directly from the hypothesis features, leading to fewer distractions and improving generalization on the dev set.
- Simplification
  - By only considering the hypothesis, the task might have become simplified in certain cases, as the model no longer needs to compare the premise with the hypothesis. In cases where the hypothesis alone provides strong clues for classification, the simplification might have led to better performance.
- Bias
  - The classifier might have been relying too heavily on the hypothesis in the first place. Many examples of entailment, contradiction, or neutral relationships can be inferred directly from the hypothesis without needing much from the premise. If the classifier was already biased toward hypothesis-driven patterns, then removing the premise might not have hurt and could even have improved performance.

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.

   Ans: The condition for the Hermitian to be set true is that our cooccurrence matrix is assumed to be Hermitian (symmetric if real-valued) in a scenario like this we should check first if the cooccurrence matrix what we are using is symmetric or not, before setting it as True or False, in our case the matrix is symmetric and real-valued. Hence we can set Hermitian as true.

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 is that the case?

Ans: Looking at the top-k similar words for each target word based on the word embeddings generated, some observations are:
This result likely indicates that frequent function words like and, the, of, to and punctuation symbols dominate the embeddings learned from the corpus. This happens because these words occur very frequently and are present in many contexts, causing their co-occurrence counts to be disproportionately high compared to meaningful content words like man, woman, happy, etc.).

High Frequency of Function Words like "the," "and," "in," and punctuation marks appear very frequently in most texts, including "Emma." These words are often context-neutral, yet because they co-occur with almost every other word, they dominate the co-occurrence matrix. Since the co-occurrence matrix captures how often words appear together in a context window, frequent words have high co-occurrence counts with many other words. As a result, SVD identifies these frequent words as having strong embeddings that dominate the similarities.

Lack of Preprocessing in the input is reflected in the output which, suggests that stop words and punctuation were not removed during preprocessing. In natural language processing tasks, it is common to remove these words because they do not carry much semantic content. Without removing stop words or applying weighting techniques like TF-IDF, frequent words dominate the co-occurrence matrix, as seen in these results.

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.

Ans:
Top-k most similar words (adjectives) to "man": Charming, pert, gallant, weak, fine
Top-k most similar words (adjectives) to "woman": Charming, pert, blush, fine, sweet, minute

What makes sense: It makes sense that charming is associated with both "man" and "woman," as it can be used to describe an appealing personality for either gender. Bias in the dataset makes the model biased as well, words like “gallant” for men or “blush” or “sweet” for women fits the stereotypical narratives of masculinity and femininity particularly present in older texts.

What is surprising: ‘weak’ for men and ‘minute’ for women both these terms don’t fit into the gender stereotypes as men are often related with strength and ‘minute’ is somewhat an out of place adjective.

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?

   Ans:

Top-k most similar words to "happy": distressing, agreeable, superior, sorry, likely, easy, unpleasant, interesting, inferior, pleasant
Top-k most similar words to "sad": little, different, few, quivering, lame, momentary, large, sudden, small, delightful

I would expect more synonyms related to "happy, but the list shows a mix of both synonyms and antonyms. The presence of opposites like "distressing" and "unpleasant" could indicate that these words appear in contrasting contexts, which results in their co-occurrence in the corpus. The words are more varied and seem unrelated to the emotion "sad." Words like little, different, few, momentary, large, and small are more quantitative or descriptive rather than emotional. There is one obvious antonym: delightful, which directly contrasts "sad." I would expect more words directly related to "sadness". However, this list includes more neutral descriptors and antonyms, suggesting the model may not have captured the emotional dimension as well for "sad."