CS4740: Intro to NLP
Project1: Language Modeling and Word Embeddings

Part 1: Due Monday, Sept 11, 11:59PM
Part 2: Due Thursday, Sept 21, 11:59PM

1 Overall Goal

In this project we will build an **n-gram-based language model** for **sentiment classification** and also investigate a different representation for words using **word embeddings**.

The project is divided into parts. In Part 1, we will get started with the steps towards building the language model using a sentiment dataset. In Part 2, you will (a) use this language model for the sentiment classification task and (b) EITHER intrinsically evaluate word embeddings using a word analogy task OR extrinsically evaluate them on sentiment classification.

Logistics: You should work in groups of 2 or 3 students. Students in the same group will get the same grade. Thus, you should make sure that everyone in your group contributes to the project. Also, **remember to form groups on CMS**. You can use Piazza for finding your project partners.

Advice: The report is important! The report is where you get to show that you understand not only *what* you are doing but also *why* and *how* you are doing it. So be clear, organized and concise; avoid vagueness and excess verbiage.

2 Unsmoothed n-grams

To start, you will write a program that computes **unsmoothed unigram and bigram probabilities** from an arbitrary corpus. You can use any programming language(s) you like.

Assume that you are given raw (ascii) text as input (see 2.1). You may need to do tokenization, based on the design decisions you make. You may use existing tools just for the purpose of preprocessing (e.g. word tokenization) but you must write the code for gathering n-gram counts and computing n-gram probabilities yourself. For example, consider the simple corpus consisting of the sole sentence:

the students liked the assignment

Part of what your program would compute for a bigram model, for example, would be the following:

$$\begin{aligned}&P(the) = 0.4\\&P(liked) = 0.2\\&P(the|liked) = 1.0\\&P(students|the) = 0.5\end{aligned}$$

2.1 Dataset

You are given (via CMS) a sentiment corpus [1] that contains movie review excerpts from the `rottentomatoes.com` website. In the SENTIMENTDATASET folder you will find folders including training corpora for positive and negative sentiment, respectively. **You should consider positive sentiment and negative sentiment as separate corpora and generate a language model for each.**

2.2 Preprocessing

The `pos.txt` and `neg.txt` files included in each of the subfolders in the SENTIMENTDATASET folder are already tokenized and hence it should be straightforward to obtain the tokens by using space as the delimiter. Feel free to do any other preprocessing that you might think is important for this corpus. **Do not forget to describe and explain your pre-processing choices in your report.**

3 Random sentence generation

Next, you will write code to generate random sentences based on your unigram and bigram language model(s). Develop a random sentence generator for each `training` corpus (i.e. for the training corpus associated with each sentiment). It should produce sentences from scratch as described in class. Also, experiment with seeding — start from an incomplete sentence of your choice and use your language model to complete it (instead of generating from scratch). **Include examples of the generated sentences from each language model in your report. Analyze the generated sentences and compare the sentences generated with unigram vs. the bigram models.**

The work from Sections 2 and 3 constitute **PART ONE** of the project. See Section 10 for a summary of what to submit for this part of the project.

[1] `https://nlp.stanford.edu/sentiment/index.html`

4 Smoothing and unknown words

For the sentiment classification task, you will need to implement smoothing and and a method to handle unknown words in the test data. Teams can choose any method that they want for each. **The report should make clear what methods were selected**, providing a description if the approach is non-standard, e.g., was not covered in class or in the readings.

5 Perplexity

Implement code to compute the perplexity of a test set. **Compute and report the perplexity of each of the language models (trained on positive and negative sentiment corpus) on the `development corpora`.** Compute perplexity as follows:

$$PP = \left(\prod_{i}^{N} \frac{1}{P(W_i|W_{i-1}, \ldots, W_{i-n+1})} \right)^{\frac{1}{N}}$$
$$= \exp \frac{1}{N} \sum_{i}^{N} - \log P(W_i|W_{i-1}, \ldots, W_{i-n+1})$$

where N is the total number of tokens in the test corpus and P($W_i|W_{i-1}, \ldots, W_{i-n+1}$) is the n-gram probability of your model. Under the second definition above, perplexity is a function of the average (per-word) log probability: use this to avoid numerical errors.

6 Sentiment classification

In this part of the project, you will use language models to automatically predict the most likely sentiment of a new, unlabeled movie review.

Hint. There are potentially many ways to use language models as predictors. The phrasing in the very first sentence in this section gives away one approach that is probabilistically meaningful. You may explore other ways as well; regardless, **make sure to clearly explain your method, and why you chose it, in the report**. We don't recommend that you try fancy or advanced ideas without first implementing and evaluating the simple, straightforward one we intended. (Good life advice in general!)

Methodology suggestion. You are given plenty of labeled movie excerpts for both types of sentiment (positive, negative). To adjust your models for best performance (e.g. tuning smoothing parameters, or choosing between design options) you should use the data files in the `development set` to evaluate the performance of your model. Use the folder structure to determine the `training` and `test` sets as well as the class (sentiment) labels (e.g. all the excerpts in `train/pos.txt` are training excerpts with positive sentiment).You should use only the training data and development data for developing your model. **Submit your predictions for the excerpts in the test data on Kaggle.** See Section 6.1 below for the format. Keep in mind that the evaluation metric used on Kaggle is accuracy — the number of correct predictions divided by the number of excerpts in the test set.

6.1 Kaggle submission format

Apart from your report, you will submit a **.csv** file to Kaggle. There should be one prediction on each line for each movie review excerpt inside the test folder. The prediction values are integers using the following encoding: positive: 0; negative: 1.

To complete Project 1, choose EITHER the investigation described in Section 7 OR in Section 8.

7 Evaluating word embeddings

In this part of the assignment, you will try to evaluate the quality of pre-trained word embeddings (Section 7.2). Word embeddings are often described as a distributed meaning representation for words. Most methods for deriving word embeddings/vectors are based on the general idea that words that occur in similar contexts are similar in meaning and should have similar vector representations. Word embeddings are trained on large collection of text without using a specific NLP task.

One method proposed to evaluate the quality of word embeddings is the **word analogy task** described in class (e.g., *man* is to *woman* as *king* is to *queen*). In particular, given a pair of words ⟨ a, b ⟩ and another word c, a word analogy system should find the word d such that c and d are related in the same way as a is related to b. For the purpose of this assignment, we need not identify the type of relationship.

Mikolov et al. [2] proposed vector operations on word embeddings for this analogy task. Let $\mathbf{v}_a$ be the vector representation for a, $\mathbf{v}_b$ for b. Then, in order to find the word d such that the analogy holds, we expect

$$\mathbf{v}_b - \mathbf{v}_a \approx \mathbf{v}_d - \mathbf{v}_c \tag{1}$$

$$d = \arg \max_{d \in V \setminus \{a,b,c\}} \cos(\mathbf{v}_d, \mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c) \tag{2}$$

Thus, you can compute cosine similarity with respect to all the words in the vocabulary (V) and then report the word with the highest cosine similarity.

7.1 Dataset

You have been given a subset of the data for Mikolov's analogy task in the file `analogy_test.txt`, which includes four types of semantic relations (past-tense, adjective-to-adverb, comparative, plural) and another four types of syntactic relations (city-in-state, currency, capital-common-countries, nationality-adjective). In the test file, each line is a analogy question, for example:

Athens Greece Oslo Norway

Thus, your cosine similarity metric will take the first three words as input and determine the fourth word. And your answer will be considered correct if the word you predict matches the fourth word in the analogy.

7.2 Pre-trained word embeddings

Pre-trained word embeddings are word embeddings that have already been trained on a large corpus and can be directly used in any NLP task. For this assignment, you can use any pretrained embeddings you can find. Remember to cite the paper and mention the choice of the word embeddings in your report.

Some of the pretrained embeddings that are popular are :

- Word2vec [2] `https://code.google.com/archive/p/word2vec/`
- Glove [3] `https://nlp.stanford.edu/projects/glove/`
- Dependency-based embeddings [1] `https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/`

7.3 Your task

- Pick any **two** sets of pre-trained word embeddings (see 7.2) and then compare their performance using the analogy test dataset provided with the assignment. **The report should include a table of the numeric results as well as a discussion of the results.**
- Design two new types of analogy tasks with three examples of each type. You will create your own test questions and report the performance of the choice of word embeddings on those tasks. **The report should describe the analogy tasks, include the three examples of each, and include a table and discussion of the results.**
- One of the known problems with word embeddings is the way they handle antonyms — both the word and its antonyms tend to have similar word embeddings. Verify this by finding the top 10 most similar words (using the cosine metric) for a few verbs such as *enter* or *increase*. **Report on and explain your findings.**

8 Sentiment classification with word embeddings

This part of the assignment asks you to use pre-trained word embeddings (of your choice) to perform sentiment classification of the same movie review excerpts used earlier in this project. Given a movie review excerpt, you want to determine its sentiment (positive or negative). For this purpose, you will use the word embeddings to compute a vector representation for the excerpt and compare it to a similarly computed vector representation for the positive reviews and negative reviews. **If the excerpt representation is closer (via the cosine similarity metric) to the positive vs. negative reviews, then return 'positive' as the predicted sentiment; return 'negative' otherwise.**

To compute the excerpt and review set representations, you will experiment with a very simple model that **simply averages the word embeddings of all the words in a given excerpt or set of reviews**.

More formally, if the movie excerpt (R) is

it was an amazing movie

then you use the pre-trained word embeddings as vector representations of its words: $\mathbf{v}_{it}$, $\mathbf{v}_{was}$, $\mathbf{v}_{an}$, $\mathbf{v}_{amazing}$, $\mathbf{v}_{movie}$. Next, we compute the representation (x_R) of the movie excerpt R, by averaging the word vector representations:

$$x_R = (\mathbf{v}_{it} + \mathbf{v}_{was} + \mathbf{v}_{an} + \mathbf{v}_{amazing} + \mathbf{v}_{movie})/5. \tag{3}$$

To represent the **set** of positive (or negative) reviews, you can use the above approach to obtain a vector representation for each review, and then take the average of the review-level vectors.

For submission, **run your model on the files in the test folder and submit your predictions on Kaggle**.

8.1 A machine-learning variation

THIS OPTION ONLY MAKES SENSE FOR TEAMS WITH PRIOR MACHINE LEARNING EXPERIENCE. As an alternative to the above, you can use the word embedding representation of the review excerpts (from the training as well as the test set) as **features** for any classifier of your choice.

If you were to use a linear classifier (you can use any existing implementation) then you can learn the weights (w) associated with each feature such that the classifier learns to predict the correct label (y).

$$y = \mathbf{w}.\mathbf{x} \tag{4}$$

Like the language-model-based classifier from Section 6, train your model on the examples in the train folder and tune your model using the development set.

For submission, **run your model on the files in the test folder and submit your predictions on Kaggle**.

9 Grading rubric

Part One:

- Unigram and bigram probability table; random sentence generator (10%)

Part Two:

- Smoothing, unknown word handling, and perplexity calculation (10%)
- Implementation of perplexity (10%)
- Sentiment classification with language models (10%)
- Evaluating word embeddings *or* Sentiment classification with word embeddings (20%)
- Experiment design and methodology; report quality (35%)
- Kaggle submission (5%)

Note: You may make up some of the lost points in Part One if you submit a corrected version together with Part Two. (Maximum points you can make up this way is 7/10 for Part One).

Performance concerns: While you shouldn't focus on optimization, nothing in this assignment should need to be slow. If you find your implementations taking really long amounts of time, something is probably wrong with your design. You may lose points for impractically slow, brute-force style solutions, where applicable.

10 What to submit

For Part One (via CMS and Gradescope): Submit a zip file with your code. Also submit examples of the random sentences produced and the analysis of the generated sentences (in a pdf file).

For Part Two (via CMS and Gradescope): Check the submission format for each of the tasks as explained above. Submit your (1) report (a .pdf) and (2) code on CMS as a .zip or .tar file. Also submit (3) the csv files produced for Section 6 and Section 8 on Kaggle. We will have separate Kaggle challenges for the two submissions. We will notify about the submission link later via piazza.

In order to be graded as a group, **you must form groups with your partner(s) on Kaggle.** You will have a limited number of Kaggle submissions per group.

11 The report

With the Part Two submission, you should submit a short document (no more than 6 pages) that describes your work. Your report should contain **a section for each of the Sections 2-8 above** as well as **a short *workflow* section** that explains how you divided up the work for the project among group members. **Important:** the role of this document is to show us that you **understand** and **are able to communicate** what you did, and how and why you did it. **Come see us if you're not sure how to answer either of these questions!**

You might think this means you must write a lot to explain something, it may be a bad sign. Describe the general approach, the data structures used (where relevant). Use examples, identify all design choices and justify them (e.g., how did you deal with unknown words? how did you perform smoothing?). Wherever possible, **quantify your performance** with evaluation metrics (e.g., report classification performance on the development set). If you tried any extensions, perform measurable experiments to show whether or not your extensions had the desired effect? If something doesn't work or performs surprisingly, try to look deeper and explain why.

Code. You may include snippets of your code in the report, if you think it makes things clearer. Include **only** relevant portions of the code (such as a few lines from a function that accomplish the task that you are describing a nearby paragraph, to help us understand your implementation). Including boilerplate code or tedious, unnecessary blocks (e.g., reading files) only makes your report cluttered and hard to follow, so avoid it.

References

[1] Omer Levy and Yoav Goldberg. Dependencybased word embeddings. In *In ACL*, 2014.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.