Assignment 1

COMP3361: Natural Language Processing - University of Hong Kong

Spring 2025

Reading Materials: We **highly recommend** that you read J&M Ch. 3, J&M Ch. 6, and M. Collins, Notes 1 before doing this assignment.

Resources: You can access all related resources at `https://github.com/ranpox/comp3361-spring2025/tree/main/assignments/A1/`.

Code: We provide a Jupyter Notebook template `https://github.com/ranpox/comp3361-spring2025/blob/main/assignments/A1/A1.ipynb`. You can use it on your local environment or Google Colab (recommended) `https://colab.research.google.com/github/ranpox/comp3361-spring2025/blob/main/assignments/A1/A1.ipynb` to finish it. Some useful packages you may need: `nltk`, `numpy`, `scikit-learn`

Grade: We will evaluate your grade based on the code quality, output results/log, and discussion. We don't evaluate your submission by perplexity in Section 1, but better prediction accuracy in Section 2 increases your score. There are some optional problems in this assignment, and solving them increases your grade. Please don't waste your time **manually** searching for hyperparameters. You are more than welcome to introduce a new hyperparameter optimization method to find more reasonable values.

Submit: You should submit the **UniversityNumber.ipynb** file and final prediction file **UniversityNumber.test.txt** of Section 3.2 to moodle. The prediction file must contain one label (0 or 1) per line in the same order as test-blind.txt, for example:

```
0
1
0
...
```

All code and discussion should be included in your submitted `.ipynb` file. Make sure your code does not use your local files so that the results are reproducible. Before submitting, please **1. clean all outputs and 2. run all cells in your notebook and keep all running logs** so that we can check the results.

1 n-gram Language Model (45%)

In this section, you will build your own language model. We provide a dataset with three files containing many whitespace-tokenized sentences at `https://github.com/ranpox/comp3361-spring2025/tree/main/assignments/A1/data/lm`:

- `train.txt`: data for training your language model.
- `dev.txt`: data for developing and optimizing your language model.
- `test.txt`: data for only evaluating your language model.

You will first build vocabulary with the training data, and then build your own unigram, bigram, and trigram language models with some smoothing methods.

1.1 Building Vocabulary

You will download and preprocess the tokenized training data to build the vocabulary. To handle out-of-vocabulary(OOV) words, you will convert tokens that occur less than three times in the training data into a special unknown token `⟨UNK⟩`. You should also add start-of-sentence tokens `⟨START⟩` and end-of-sentence `⟨END⟩` tokens. If you did this correctly, your language model's vocabulary, including the `⟨START⟩`, `⟨END⟩` and `⟨UNK⟩` tokens should have 24,067 words.

Please show the vocabulary size and discuss the number of parameters of n-gram models.

1.2 n-gram Language Modeling

After preparing your vocabulary, you are expected to build bigram and unigram language models and report their perplexity on the training set, and dev set. Please discuss your experimental results. If you encounter any problems, please analyze them and explain why.

1.3 Smoothing

In this section, you will implement two smoothing methods to improve your language models.

1.3.1 Add-one (Laplace) smoothing

Please improve your bigram language model with add-one smoothing. Report its perplexity on the training set and dev set. Briefly discuss the differences in results between this and the bi-gram model you built in Section 1.2.

1.3.2 Add-k smoothing.

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (such as 0.5, 0.05, or 0.01). This algorithm is therefore called add-k smoothing.

$$P^*_{\text{Add-k}}\left(w_n \mid w_{n-1}\right) = \frac{C\left(w_{n-1}w_n\right) + k}{C\left(w_{n-1}\right) + kV} \tag{1}$$

Please optimize the perplexity on the dev set by trying different k (no more than three times). Report its perplexity on the training set and dev set. Briefly discuss the differences in results between this and the bi-gram model with add-one smoothing.

1.3.3 Linear interpolation

we always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.

$$\hat{P}\left(w_n \mid w_{n-2}w_{n-1}\right) = \lambda_1 P\left(w_n\right) + \lambda_2 P\left(w_n \mid w_{n-1}\right) + \lambda_3 P\left(w_n \mid w_{n-2}w_{n-1}\right) \tag{2}$$

Please implement linear interpolation smoothing between unigram, bigram, and trigram models. Report its perplexity on the training set and dev set for the values $\lambda_1 = 0.2, \lambda_2 = 0.3, \lambda_3 = 0.5$. Please optimize on a dev set by trying different hyperparameter sets. Finally, report the perplexity on the test set with the best hyperparameter set you get. Briefly discuss the results.

1.3.4 Optimization

So far, we have manually chosen the hyperparameter that minimizes the perplexity of the dev set and evaluated it on the test set. There are various ways to find this optimal set of hyperparameters. Do you know any other learning algorithms? Please give an example.

2 Word Vectors (20%)

In this section, you will explore GloVe word vectors.

2.1 Find most similar word

Use cosine similarity to find the most similar word to each of these words. Report the most similar word and its cosine similarity.

- `ubiquitous`
- `serendipity`
- `melancholy`
- `paradox`
- `ethereal`
- `cacophony`

2.2 Finding analogies

Use vector addition and subtraction to compute target vectors for the analogies below. After computing each target vector, find the top three candidates by cosine similarity. Report the candidates and their similarities to the target vector.

- `king : queen :: man : ?`:
- `paris : france :: rome : ?`:
- `france : french :: germany : ?`:
- `london : england :: paris : ?`
- `cat : kitten :: dog : ?`
- `rich : poor :: strong : ?`

3 Sentiment analysis (35%)

In this section, you'll be using the movie review dataset to evaluate word vectors. You are provided with three files:

- `train.txt`: Your labeled training data, you should and can only train on this.
- `dev.txt`: Your labeled development data.
- `test-blind.txt`: On the blind test set, you do not see the labels and only the sentences are given to you.

3.1 Logistic Regression (25%)

In this section, you should build three types of features to train a classifier: Unigram, Bigram, and GloVe (Global Vectors for Word Representation) features. Use logistic regression to predict the sentiment labels of movie reviews. You can only use train.txt to train the model. Report the P(precision), R(recall), F1 results of dev.txt in a table and compare the performance difference of these three features.

Note: For GloVe features, handle out-of-vocabulary words by either:

- *Using a zero vector*
- *Averaging available word vectors in the sentence*

3.2 Better Feature (10% + Extra 5% for outstanding solutions)

Try at least one feature modification or another ML algorithm (although either should work). Report the performance of dev.txt it gives. Things you might try: other types of n-grams, tf-idf weighting, clipping your word frequencies, discarding rare words, discarding stopwords, etc. After selecting your best method, make predictions on test-blind.txt and save it as **UniversityNumber.test.txt**. The saved format should be identical to dev.txt, and the order of examples should also be identical to test-blind.txt.