

This assignment aims to use n-gram based language model to write a program that will classify authors based on a training text. Please carefully read [Section 3.5 of Jurafsky and Martin](#) and [NLTK's LM package](#) before you start. The lead TA for this assignment is Shirley Anugrah Hayati ([hayat023@umn.edu](mailto:hayat023@umn.edu)). Please communicate with the lead TA via Slack, email, or during office hours.

## Setup

I have created four source files containing excerpts from multiple works by different authors: Jane Austen, Charles Dickens, Leo Tolstoy, and Oscar Wilde. You can find these files in [this link](#). Download the files from the folder. You will need to decide which encoding you want to use. I have posted the UTF-8 encoded text and the ASCII “transliteration” of the UTF-8 encodings. Some peculiar things are going on; in particular, look at the first few paragraphs of Tolstoy’s text to see some examples of the differences in encodings.

```
ASCII possible. Kut'uzov himself with all his transport took the ..
UTF8 possible. Kutúzov himself with all his transport took the ..
```

Note that UTF-8 encoding is **recommended in most cases** as most text editors, websites, text data on the Internet, and many programming languages use UTF-8 by default.

Your task is to write a program that will build a language model for each author and attempt to map a new text to the author who originally wrote it using the language model.

## Your Task

Write a program `classifier.py` that can be run with the following command-line setups:

```
python3 classifier.py authorlist

python3 classifier.py authorlist -test testfile
```

Where *authorlist* is a file containing a list of file names like the following:

```
austen.txt
dickens.txt
tolstoy.txt
wilde.txt
```

That gives the file names of the training sets that you will use.

When the program is run without the `-test` flag, your program should automatically extract a development set (10%) from the author data, train on the remaining training data, and run the task on the development data and print out the results.

When the program is run with the `-test` flag, your program should use the entirety of data in each author file to train a language model, then output classification results for each line in the given file *testfile*. You may assume that each line of *testfile* is an entire sentence.

## Sample Runs

```
$ python3 classifier.py authorlist
splitting into training and development...
training LMs... (this may take a while)
Results on dev set:
austen      61.4% correct
dickens     73.3% correct
tolstoy     57.7% correct
wilde       67.3% correct
```

```
$ python3 classifier.py authorlist -test test_sents.txt
training LMs... (this may take a while)
austen (the first line of text is predicted as austen)
austen
wilde
austen
tolstoy
...
```

Your classifier should proceed as follows:

- For each data set, create an n-gram language model using [NLTK's LM package](#) as a baseline model.
- Improve your n-gram language model (i.e., reduce perplexity) by using different types of smoothing, backoff, and interpolation. Carefully read [Section 3.5 of Jurafsky and Martin](#) and use default functions implemented in NLTK: [smoothing](#), [backoff](#), and [interpolation](#). Feel free to try a few different combinations of models (e.g., uni+bigram model with smoothing) and see which works the best for this task.
- For each of your language models, compute the perplexity of the test item. Whichever language model gives the lowest perplexity should be how you classify the test item.
- For each of your language models, generate five samples of each author given the same prompt you specify and compare them.
- (optional, bonus point +2) Implement ngram language models without using NLTK. You can implement it using Numpy or PyTorch from scratch. For instance, this [tutorial](#) and [code](#) by Andrej Karpathy describes how to build GPT language model using PyTorch, but you will only see the video frames from [7:52](#) to [42:12](#) for bigram language model implementation.

General Instructions: Please present your results using tables whenever appropriate. Please try to avoid copying and pasting sentences from sources.

Below is some basic code to preprocess the training data, train N-gram language models, and calculate perplexity, taken from the [NLTK's LM package](#) tutorial. Carefully read the detailed process in the tutorial.

```

% Prepare Data
% Note: You may use other libraries for tokenization and sentence segmentation
>>> from nltk.lm.preprocessing import pad_both_ends, flatten, padded_everygram_pipeline
>>> list(pad_both_ends(text[0], n=2))
['<s>', 'a', 'b', 'c', '</s>']
>>> list(bigrams(pad_both_ends(text[0], n=2)))
[('<s>', 'a'), ('a', 'b'), ('b', 'c'), ('c', '</s>')]
>>> list(flatten(pad_both_ends(sent, n=2) for sent in text))
['<s>', 'a', 'b', 'c', '</s>', '<s>', 'a', 'c', 'd', 'c', 'e', 'f', '</s>']
>>> train, vocab = padded_everygram_pipeline(2, text)

% Training
>>> from nltk.lm import MLE
>>> lm = MLE(2)
>>> lm.fit(train, vocab)
>>> print(lm.vocab)
<Vocabulary with cutoff=1 unk_label='<UNK>' and 9 items>
>>> len(lm.vocab)
9
>>> lm.vocab.lookup(text[0])
('a', 'b', 'c')
>>> lm.vocab.lookup(["aliens", "from", "Mars"])
('<UNK>', '<UNK>', '<UNK>')

% Inference
>>> test = [('a', 'b'), ('c', 'd')]
>>> lm.perplexity(test)
2.449489742783178

```

## Deliverables

Please upload your code and report to [Canvas](#) by **Oct 15, 11:59pm**.

**Code:** You should provide a zipped file containing your training/inference scripts or a link to your github repository.

**Report:** Maximum four pages PDF. Your report needs to include the following content:

- What encoding type your program runs on
- What information is in your Language Models (bigrams, trigrams, etc)
- What method of smoothing you are using
- How do you deal with out-of-vocabulary words during run time when you build a language model?
- Any other tweaks you made to improve results (backoff, etc.)
- The results (i.e., accuracy for each author) you get with the given data with an automatically-extracted development set (i.e. the output from running it without the -test flag)

- For each of your language models, generate five samples of each author given the same prompt you specify with their perplexity scores (i.e., a total of five samples and perplexity scores by four different language models).

## Rubric (50 points + 10 bonus points)

- **Basic NLTK NGram Model Construction (15 points)**
  - Full Marks
  - Data is not properly cleaned (for example, just tokenization is done): -0.5 to -1
  - Mistakes in the process of tokenizing the words/ not getting rid of new lines and empty spaces: -1 to -2 points
  - Mistakes when creating bigrams/ trigrams -1 to -2 points
  - Major mistakes when creating bigrams/ trigrams -3 to -5 points
  - Does not consider bigrams/ trigrams at all: -8
  - No Marks
- **Use of a Model with smoothing (10 points)**
  - Full Marks, the correct implementation of the smoothing algorithm
  - Minor mistakes in results or code
  - Major mistakes in results or code
  - No Marks
- **Baseline that is better than random (5 points)**
  - Full Marks
  - Algorithm performs on avg less than 25% : -1 to -2 points
  - Accuracy results not reported (e.g. showing only perplexity instead of accuracy): -3 to -4
  - No Marks, Code doesn't run
- **Report (10 points)**
  - Full Marks, Report contains full information about how to run the program, tokenization, smoothing, your language models, and your results
  - Partial, Some pieces of report are missing
  - No Marks, No report
- **Training/Dev set construction (10 points)**
  - Full Marks, automatically creates dev set and runs on it
  - Minor mistakes in dev set creation
  - Major mistakes in dev set creation
  - No Marks
- **Bonus Point (Max: +10 points)**
  - Implement n-gram language models without using NLTK library, +5 points
  - Show the best result (i.e., the lowest perplexity in the class, the highest accuracy), +1 to +2 points
  - Show results and analyses on various n-gram models/smoothing methods +1 to +3 points