PA2 Autocorrect: Instructions

Help Center

Homework 2: Autocorrect [starter code]

Correcting Spelling Errors

Due: Tuesday March 27 at 5:00pm Pacific Daylight Time

In this assignment you will be training a language model to build a spell checker. Specifically, you will be implementing part of a noisy-channel model for spelling correction. We will give the likelihood term, or **edit model**, and your job is to make a **language model**, the prior distribution in the noisy channel model. At test time you will be given a sentence with exactly one typing error. We then select the correction which gets highest likelihood under the noisy-channel model, using your language model as the prior. Your language models will be evaluated for accuracy, the number of valid corrections, divided by the number of test sentences.

Data

We will be using the writings of secondary-school children, collected by David Holbrook. Note that this corpus is available online, so do not Google around for it! The training data is located in the data/ directory. A summary of the contents:

- **holbrook-tagged-train.dat:** the corpus to train your language models
- **holbrook-tagged-dev.dat:** a corpus of spelling errors for development
- **count_1edit.txt :** a table listing counts of edits $x|w$, taken from Wikipedia. You don't need to modify any code which uses this.

Note that the data files do not contain <s> and </s> markers, but the code which reads in the data adds them.

Your Assignment, Should You Choose to Accept It

Implement the following language models:

- **Laplace Unigram Language Model:** a unigram model with add-one smoothing. Treat out-of-vocabulary items as a word which was seen zero times in training.
- **Laplace Bigram Language Model:** a bigram model with add-one smoothing.
- **Stupid Backoff Language Model:** use an unsmoothed bigram model combined with backoff to an add-one smoothed unigram model
- **Custom Language Model:** implement a language model of your choice. Ideas include interpolated Kneser-Ney, Good-Turing, linear interpolated, trigram, or any other language model you can come up with. You should not train your models on different training data than supplied. Your goal is for your custom language model to perform better than any of the other three language models we ask you to implement.

We have provided you with a uniform language model so you can see the basic layout, located in UniformLanguageModel.{py,java}.

To implement a language model you need to implement two functions:

- **train(HolbrookCorpus corpus):** takes a corpus and trains your language model. Compute any counts or other corpus statistics in this function. See the example UniformLanguageModel for how to access sentences in the corpus and the words in those sentences.
- **score(List words):** takes a list of strings as argument and returns the numerical score, which should be the log-probability of the sentence using your language model. Use whatever data you computed in train() here.

Evaluation

Your language models will be evaluated on the development data set and a held-out test set. The performance of each language model will be evaluated with respect to the performance of our solution implementation. To help with your implementation, we give you the expected performance of each language model on the development set:

- **Laplace Unigram Language Model:** 0.11
- **Laplace Bigram Language Model:** 0.13
- **Stupid Backoff Language Model:** 0.18
- **Custom Language Model:** at least as good as Stupid Backoff, so 0.18.

Note that the performance we expect from your language model is not that great! We have provided you with a very simple edit model, not a lot of training data, and the task is rather difficult. You will receive full credit for implementations which meet the stated thresholds, and a linearly decaying score for accuracy less than the reference.

Given Code

The rest of the scaffolding has been provided (reading corpora, computed edit probabilities, computing the argmax correction). A short summary of the remaining files:

- **SpellCorrect.{py, java}**: Computes the most likely correction given a language model and edit model. The main() function here will load all of your language model and print performance on the development data, useful for debugging. It may be useful to comment out some of the tests in main() when developing.
- **EditModel.{py, java}**: Reads the count_1edit.txt file and computes the probability of corrections. The candidate corrections are all strings within Damerau-Levenshtein edit distance 1. The probability of no correction is set at .9 ($P(x|x) = .9$). Note that the EditModel isn't great, but your language models will be evaluated using this model, so it won't effect your grade.
- **HolbrookCorpus.{py, java}**: Reads in the corpus and generates test cases from misspellings.
- **Sentence.{py, java}**: Holds the data for a given sentence, which is a list of Datums. Contains helper functions for generating the correct sentence and the sentence with the spelling error.
- **Datum.{py, java}**: Contains two strings, word and error. The word is the corrected word, and error contains the spelling error. For tokens which are spelled correctly in the corpus, error = ''.

Running the code

For Java, execute

```
$ cd java
$ mkdir classes
```

```
$ javac -d classes -target 1.6 *.java
$ java -cp classes SpellCorrect
```

This will train all of the language models and output their performance.
Similarly, for python, execute

```
$ cd python
$ python SpellCorrect.py
```

Submitting Your Solution

For each language, we have provided a submission script/program which will run your program and autocorrect a given corpus. This data will then be sent to our servers where it will be stored and graded once the assignment is due. For each language, the submission program will programmatically call your script and then prompt you for your username and a password. The password can be found at the top of the Assignments page on this website. It will also ask you about which parts of the assignment you wish to submit. For this assignment your system will be evaluated in eight parts: each language model will be evaluated on both the development data and a held-out test set.
Here are the basic shell commands which you will need to execute for Java:

```
$ cd java
$ mkdir classes
$ javac -d classes *.java
$ java -cp classes Submit
```

and for Python

```
$ cd python
$ python submit.py
```

It is important that these scripts be executed from either the java or python directories so that they can easily import the code you've written and so that the relative path "../data/" correctly points to the training and development data (it is hard-coded into the submission scripts).