ECS 189G
Homework 1

By Zhou Yu and Dan Jurafsky
Updated by Kevin Jesse

January 16, 2018

Description

Code

Homework 1 Code.

Correcting Spelling Errors

This assignment will implement a noisy-channel model for spelling correction. This means creating an **edit model** (likelihood terms) and a **language model** (distribution in the noise channel model). At test time, a sentence with exactly one error will be passed into your edit model which will perform various edits and select the correction that gives the highest likelihood under the language model. Your language model will finally be evaluated for accuracy over valid corrections, divided by the number of sentences tested.

Data

Training and testing will be on a misspelling tagged corpus by David Holbrook.

- **train.dat**: the corpus to train your language models
- **dev.dat**: a corpus of spelling errors for development
- **count_1edit.txt** : a table listing counts of edits taken from Wikipedia.

Note when reading sentences into our models, <**s**> and </**s**> markers denoting the beginning and the end of the sentence, are inserted and sentences in the corpus do not contain these markers.

1 Edit Model

Lets begin by creating the backbone of the autocorrect, the **edit model**. Your changes will occur inside of EditModel.py Given a word, you will be performing a list of 1-distance edits: deletions, insertions, replacements, and transposes. Refer to the noisy channel model slides. This list will be used by the method editProbabilities to distribute probability mass based on the table of edit counts from Wikipedia **count1_edit.txt**

What You Need To Do

Implement the following methods.

- **insertEdits**: all edits formed by inserting a character into the word.
- **transposeEdits**: all edits formed by transposing two characters in the word.
- **replaceEdits**: all edits formed by replacing a character with another.

We provided deleteEdits which demonstrates how to construct a set of candidate edits (new words) by deleting a character from original word. You will be implementing a similar concept with insertions, transposes, and replacements. Note the use of ¡ when there is no previous character to condition on; for example, a deletion of the first character. Make sure you run the unit tests on your functions before moving on. This can be done by running main in EditModel.py. Make sure your unit tests have 100% overlap.

2 Spelling Correction

Now that weve implemented the edit model, we can use it within the spelling corrector SpellCorrect.py. The method correctSentence takes in a sentence with an error, and tests each of these candidate edits from the edit model. This method iterates through all candidate edits for each word in the sentence and attributes a score of the new sentence with the probability of this edit occurring. In SpellCorrect.py main each language model is evaluated. We have provided two simple language models: uniform and unigram. We will improve on these later by creating a smooth unigram, smooth bigram, backoff smooth bigram, and a custom model of your choice.

What You Need To Do

Construct the method correctSentence as described above by using the edit model and the instantiated language model and return the most likely correct sentence. For reference, we have included the accuracies for the unigram and uniform model in the evaluation section. Their accuracies are quite low with plenty of room for improvement!

3 Extended Language Models

In this section, we will use various language models to get better results.

What You Need To Do

Implement the following language models.

- **Smoothed Unigram Language Model**: a unigram model with add-one smoothing otherwise known as a Laplace unigram language model.
- **Smoothed Bigram Language Model**: a bigram model with add-one smoothing.
- **Backoff Language Model**: use an unsmoothed bigram model with backoff. The backoff function will be a smoothed unigram model.
- **Your choice of a language model**: ideas can be interpolated Kneser-Ney, linear interpolated, trigram, or any other model. Train and test with the same data supplied to other models.

Use UnigramModel.py as an example for how to create your train and score functions for each model. Note, treat all unknown words not in your trained model as a single word UNK that appears in the training vocabulary with a frequency of 0. This allows for your trained model to handle words it may not have seen before.

All language models have two functions: train and score.

- **train(Corpus c)**: This function takes in a data corpus, iterates through each sentence and word and computes the counts of their occurrences or any other statistics necessary for the specific model. Note a datum consists of a word and error pair.
- **score(List sentence)**: This function takes a list of strings that compose the sentence and returns the probability of the sentence with the candidate edit from your correctSentence function. Rather than traditionally multiplying all probabilities together, it is important to accumulate the sum of their logs (equivalent) due to underflow problems. Use whatever data from the train method here.

Evaluating The Language Models

The language models will be evaluated on the dev data set and test set. The performance will be evaluated with our solution implementation. Here is the expected performance of the following language models on the dev data set.

- Smooth Unigram Model: .11
- Smooth Bigram Model: .13
- Backoff Model: .18
- Your Language Model of choice: .01 better than backoff model

You will get full points for models that score within .005 or above these accuracy goals. Models that achieve above 70% of this goal will get half points. Notice the performance metrics are not great; this can be from a few factors but significantly because we lack a lot of training data and the task is quite difficult.

Running The Code

- To sanity check your edit model, run `$ python EditModel.py`
- To verify performance of language models, run `$ python SpellCorrect.py`

Submitting to Canvas

Please archive the following in a .tar.gz or .tgz (other formats not accepted) labeled **hw1.tar.gz** or **hw1.tgz**.

- EditModel.py
- SpellCorrect.py
- SmoothUnigramModel.py
- SmoothBigramModel.py
- BackoffModel.py
- CustomModel.py

Grading

The assignment will be worth 50 points. Each model's dev and test set are worth 5 points, **editModel.py** and **spellCorrect.py** are both worth 5 points.

Model	Dev Set Score	Test Set Score
Smooth Unigram	5pts	5pts
Smooth Bigram	5pts	5pts
Backoff	5pts	5pts
Custom	5pts	5pts

We withhold the test set and test your models with this after you have turned your assignment in.

Errata

Please email Kevin Jesse for any errata in this document or code.