ECS 189G
Homework 1

April 10, 2019

Noisy Channel Model Spellchecker

In this homework, you will be creating a simple spellchecker. There are two parts to this assignment.

Code

The code packet can be found on Canvas.

1 Edit Model

The first part of creating the spellchecker is to create the Edit Model. This model is responsible for generating the "edits" as the candidates for spellchecking. To keep it simple, we will only be focusing on four types of edits: insertion, deletion, replacement, and transpose. We will limit them to one-distance edits only.

Familiarize yourself with the code in `EditModel.py`. Take a look at the `deleteEdits()`. Note the use of < when there is no previous character to condition on; for example, a deletion of the first character.

You will be implementing the remaining three functions: `insertEdits()`, `transposeEdits()`, `replaceEdits()`.

These four functions will generate a list of possible edits for your sentences. In the `editProbabilities()` function, each of the edits are weighed using the distribute probability mass from Wikipedia, which is provided to you in `count1_edit.txt`.

Make sure to run the unit tests provided to you in the same script. Your unit tests should have a 100% overlap.

2 Spelling Correction

The second part of creating the spellchecker is to correct the spelling. From the previous section, you should have a list of possible edits and their associated probabilities. This part finds the best edits using the probabilities and several language models you are going to create.

Familiarize yourself with the code in `SpellCorrect.py`. Also take a look at the models we provided you, namely `UniformModel.py` and `UnigramModel.py`.

You will be implementing several items.

- `correctSentence()`. This function is responsible for testing each possible edits on the sentence with error and giving the edited sentence a score. This score represents the probability of that edit occurring according to your language model.
- Language Models. The two models given to you are good places to start. 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. There are four extra models you need to implement:
  - **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.

All language models have two functions: train and score.

- **train()**: 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()**: 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.

Checking your model

You can run the following scripts to check your model. You can take a look at their main() function and see what they do.

- `EditModel.py`: unit test for your edit model.
- `SpellCorrect.py`: Getting the score for your language model.

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.

Code of Conduct

Please be reminded of the code of conduct as posted on the syllabus and on the OSSJA website. You must not share any parts of your code to others within and outside of the class. This includes, but not limited to, sharing screenshots of your code, posting or hosting your code onto platforms such as GitHub, obtain materials from previous versions of the class. At the same time, you must not use any code that is not written by you in this assignment. Your codes will be screened for plagiarism. We reserve the right to take disciplinary action to any offenders, which include reporting to Judicial Affairs.