CS753 (Autumn 2017): Assignment 1

This assignment is due by **11.59 pm on August 24th**. For each additional day after August 24th until August 27th, there will be a 10% reduction in marks. The submission portal on Moodle will be closed after 11.59 pm on August 27th.

Please read the following important instructions before starting.

1. For problem 2, please download the following archive file into your working directory before starting on the tasks.
2. Your final submission should be a .tgz bundle of a directory **organized exactly** as described here. Please make sure of this. Assignment bundles that do not follow this format will not be graded. Submit the file `submission.tgz` on Moodle.

Problems

1. Probabilities (30 points)

Consider a noisy channel through which bits can be sent. When a bit is sent, it gets flipped with probability δ (independent of the other bits) and is received correctly with probability $1-\delta$. Suppose a binary string of i.i.d. bits is drawn with each bit being 0 or 1 with probability p and $1-p$, respectively, and sent through the channel.

A. Find the probability that the kth symbol is received as 0. **[4 points]**
B. Suppose we use a repetition code, where 0 is encoded as 000 and 1 as 111. The decoder outputs the majority bit on receiving a 3-bit string. For example, 001 will be decoded as a 0. Under this scheme, what is the probability of correctly decoding a 0? **[8 points]**
C. Now suppose each bit is transmitted as it is (i.e. unencoded). But, the decoder is allowed to output 0, 1 or $\perp$. The decoder is penalized 2 points for outputting the wrong bit and 1 point for outputting a $\perp$. We are interested in finding out the decoding strategy, $D:\{0,1\}\rightarrow\{0,1,\perp\}$, that minimizes the expected penalty. Suppose $\delta=\frac{1}{8}$, specify the best decoding strategy for all values of $p\in[0,1]$. **[18 points]**

2. Spelling correction system (50 points + 5 points extra credit)

The spelling correction problem can be formulated as a search through a composition of weighted finite state transducers (WFSTs), much like speech recognition. Given a misspelling (i.e. a sequence of letters) represented as a finite-state acceptor M, an edit distance transducer E mapping letter sequences to letter sequences with associated costs, a letter-to-word transducer T that maps each word to its

corresponding letter sequence and a language model acceptor G trained on a large amount of text, the most likely sequence of valid words corresponding to M is obtained by computing the shortest path in:

$$D = M \circ E \circ T^{-1} \circ G \quad (1)$$

The output labels from the shortest path in D gives us the required word sequence.

You will use the OpenFst Toolkit for this question. You can refer to the OpenFst Quick Tour for a brief introduction to the OpenFst libraries.

You will be provided with a development set (`assgmt1/dev.txt`) to evaluate your spelling correction systems during development. Your final submissions will be tested on an unseen test set. A leaderboard showing accuracies of all the submissions will be posted on Moodle.

Before getting started, please download the following archive file into your working directory and untar it:

```
tar -xvzf assgmt1.tgz
```

Each task below requires you to generate some **`files`** that will be added to your `submission/prob2/` directory.

(A) Creating M, T and G

Let us assume that each misspelling corresponds to an isolated word. The vocabulary of words for this task is given in the file, `assgmt1/words.vocab`. Each word is also accompanied by a word count, shown in `assgmt1/wordcounts.txt`, that specifies the number of times the word occurred in a text corpus; the latter was created using text from several popular books listed at Project Gutenberg. Now we have all the files needed to implement the finite state machines in Equation (1).

1. First, build the word-to-letter FST T in Equation (1). Write a script that traverses through the word list in `words.vocab` and construct an FST that maps each word to its corresponding sequence of letters. Use the vocabulary files for words (`assgmt1/words.vocab`) and letters (`assgmt1/lets.vocab`) to compile this FST into `T.fst`. Copy `T.fst` to **`submission/prob2/T.fst`**. **[3 points]**
2. Next, build the grammar FST G in Equation (1). Since we are correcting spellings of isolated words in this sub-session, G only needs to model how often a word appears in English. The file `words.vocab` contains raw counts of word occurences. Convert this to a relative count (a unigram probability) by dividing each count by the total number of words. Create a single-state FST with arcs for every word in the vocabulary

(and negative log probabilities as weights). Compile the resulting FST text file into `G.fst`. Copy `G.fst` to **`submission/prob2/G.fst`**. **[3 points]**

3. Create a script `./spell2fst.sh` that converts misspellings (listed in the third column in `assgmt1/dev.txt`) into linear-chain FSAs and store them within an output directory named `dev_wrd`). Each misspelling has an ID (say `i`) listed in the first column in `assmgt1/dev.txt`. The corresponding FSAs should be named `{i}.fsa.txt`. The script `./spell2fst.sh` should work with the following command-line arguments:

```
./spell2fst.sh --odir=dev_wrd --vocab=assgmt1/lets.vocab assgmt1/dev.txt
```

(Each FSA created for a misspelling within `dev_wrd/` would correspond to M in Equation (1).) Copy the directory `dev_wrd` into **`submission/prob2`**. **[6 points]**

(B) Creating edit distance transducers

1. Only the FST E remains to be constructed. E will have arcs mapping letter x to letter y with arc weights indicating how often we expect a letter x to be typed when the author meant y. Create an FST text file for E that allows each letter to be inserted, deleted or substituted with a uniform cost of 1. An edit distance of at most 1 is allowed by this FST. Compile the resulting FST using the vocabulary file `lets.vocab` into `E.fst`. Copy `E.fst` to **`submission/prob2/E_basic.fst`**. **[8 points]**

> **Note that one can find the Levenshtein edit distance between two spellings A and B (i.e. the minimum number of edit operations needed to convert A to B) by computing the shortest path in $A \circ E \circ B$.**

2. Now you have all the FSTs needed to build the spelling correction system specified in Equation (1). Create a script `evaluate.sh` that evaluates the accuracy of your system on the development set `assgmt1/dev.txt` by decoding the misspelling FSAs and comparing against the correct words listed in the second column of `assgmt1/dev.txt`. Copy `evaluate.sh` to **`submission/prob2/evaluate.sh`**. (If you've written any helper scripts that `evaluate.sh` calls, please add them to **`submission/prob2`** as well.) Your command should run as follows,

```
./evaluate.sh --Mdir=dev_wrd --E=E.fst --T=T.fst --G=G.fst assgmt1/dev.txt
```

and output the accuracy rate (i.e. number of correct words / total number of words in the dataset) of your system. Add this to **`submission/prob2/acc1.txt`**. **[8 points]**

3. Suppose we allow the following edit operations for a cost of $\frac{1}{2}$: xx⟶x and x⟶xx for ever letter x. Repeat task 2 above with these additional operations and record the accuracy of your system within **`submission/prob2/acc2.txt`**. Copy your updated `E.fst` to **`submission/prob2/E_dup.fst`**. **[10 points]**
4. We observe that while typing one also tends to swap adjacent letters, thus causing spelling errors. Edit `E.fst` from task 2 above in order to incorporate such letter transpositions (i.e. swapping of adjacent letters). One swap will incur an edit cost of 1. Copy the resulting `E.fst` to **`submission/prob2/E_swap.fst`**. ***Hint***: `E.fst` would no longer be a single-state FST; it will need to expand its state space in order to remember the last letter in the string.

> **Adding transposition of adjacent letters as an edit operation would give us the Damerau-Levenshtein distance between two strings.**

Evaluate again on the development set using `evaluate.sh` to see if there are any performance improvements and record the new accuracy rate within **`submission/prob2/acc3.txt`**. **[12 points]**

5. This is for extra credit. There are many more modifications that can be made to `E.fst`. What other characteristics of spelling errors can be encoded within `E.fst`? Copy the new `E.fst` to **`submission/prob2/E_extra.fst`**. ***Hint***: Currently, all edit operations are given a uniform cost. You might expect a letter to be more easily substituted by another letter that is spatially close on a keyboard as opposed to letters far away. **[5 points]**

> **When building computational systems, it is good practice to "look at the data"; go over the misspellings in your development set and see if any error patterns jump at you. There is currently no learning component in your spelling correction system, but you could imagine learning the weights of `E.fst` if you had a corpus of misspellings.**

3. EM algorithm (20 points)

A coin tossing machine behaves as follows. Internally, it has two coins -- an unbiased gold coin which comes up heads or tails with probability $1/2$ when tossed and a biased silver coin with probability $1/4$ for heads and $3/4$ for tails. When you press a button, the machine probabilistically chooses one of the two coins -- the gold coin with probability ρ and the silver coin with probability $1-\rho$. The machine then tosses the chosen coin and reports its outcome, namely heads or tails. Your goal is to find out ρ by repeatedly pressing the button and observing the outcomes of the coin tosses. In the following, let $Z \in \{G, S\}$ (for Gold and Silver) be the random variable that denotes the coin that is chosen by the machine and let $X \in \{H, T\}$ denote the outcome of the coin toss.

A. We define $\gamma_\rho(z, x) = \Pr(Z = z | X = x)$. What is $\gamma_\rho(z, x)$, as a function of ρ, for each value of $z \in \{G, S\}$ and $x \in \{H, T\}$? **[5 points]**

B. ρ can be estimated using the Expectation Maximization (EM) algorithm, from a set of observations, $x_1 \ldots x_N$ ($x_i \in \{H, T\}$). EM is an iterative algorithm which, at each round, updates the current estimate of ρ to a new estimate ρ' that maximizes the following expression:

$$\mathcal{L}_\rho(\rho') = \sum_{i=1}^{N} \sum_{z \in \{G,S\}} \gamma_\rho(z, x_i) \cdot \log \Pr(z, x_i; \rho')$$

Find

$$\arg\max_{\rho'} \mathcal{L}_\rho(\rho').$$

Your expression can be in terms of ρ, $N_H = |\{i : x_i = H\}|$ and $N_T = N - N_H$. **[15 points]**