Computational Linguistics Midterm
Jean Mark Gawron
San Diego State University
Linguistics/CS 581
March 21, 2017
DUE: April 6, 2017 (Start of class)

1 FSAs and Regular Languages

1.1 Problem 1

Draw an FSA corresponding to both of the following regular expressions (assume the alphabet is a,b):
(1.1.1) (a+b)+
(1.1.2) (bab?)+

1.2 Problem 2

(1.2.1) Draw an FSA that accepts the following language:

The set of possible social security numbers

(1.2.2) Make sure your FSA accepts the following strings
(a) 123-45-6789
(b) 030-00-0141

(1.2.3) Make sure your machine rejects the following strings:
(a) 03-000-0141
(b) 123-4N-6789

(1.2.4) You can do this using XFST, if you read the tutorial notes. In that case you will be submitting a separate image file and not a hand drawing.

(1.2.5) Is this machine non-deterministic? Why not?

2 Transducers, Spelling and Phonology

2.1 Problem 3

Compute the miminal edit distance for the following string pair:

pole spiodle

Note: Your answer **must** use the minimal edit distance algorithm and include an edit distance table of the sort presented in the textbook. Answers not satisfying these criteria will receive a score of 0, even if the distance given is correct.

2.2 Problem 4: XFST

Attention: For this part of the midterm, you will email me your xfst script (gawron@mail.sdsu.edu).

Consider the English *e*-drop spelling rule relating forms like the following:

1. realize + ed = realized
2. realize + ing = realizing
3. realize + er = realizer
4. hope + ing = hoping
5. face + ing = facing
6. flee + ing = fleeing
7. see + ing = seeing
8. die + ing = dying

Using XFST, construct a rule that handles the deletion of *e* in these cases (example output below).

You need to handle the suffixes illustrated above, *ing, ed, er*. Use the following definition:

```
define Suf [ \%+Prog+{ing} | \%+Pres3PSg:s | \%+Past:{ed} | \%+Agentive:{er} ];
```

By email, You should turn in an XFST script file. It should be readable by XFST. Answers that are not readable will receive a score of 0.

Note: a script file is loaded with the command

xfst[0]: source *filename*

This is how you test that your file is readable. When you do this you should get print outs for each of the machines defined in your script file.

```
 <<Defining Sets>>
defined Vowel: 172 bytes. 2 states, 5 arcs, 5 paths.
defined Cons: 540 bytes. 4 states, 21 arcs, 19 paths.
 <<Defining Root>>
```

```
defined Root: 500 bytes. 11 states, 11 arcs, 3 paths.
 <<Defining Suf>>
defined Suf: 396 bytes. 6 states, 9 arcs, 5 paths.
 <<Defining Lexicon>>
defined Lexicon: 780 bytes. 16 states, 25 arcs, 15 paths.
 <<Defining Rules>>
defined Rules: 4.7 Kb. 23 states, 332 arcs, Circular.
 <<Defining Final Regex>>
848 bytes. 18 states, 28 arcs, 15 paths.
Closing file my_answer.xfst...
```

You can facilitate testing your machine by preparing a text file that contains words you want to transduce. Call the text file words.txt. Here are some words you should have in your test file (and some your words your answer MUST handle):

```
hoping
realizer
realizing
realized
fleeing
seeing
dying
```

If you put that file in the same directory you start XFST in, you could do:

```
xfst[1]: apply up < words.txt
```

If your machine is correctly specified, you should se the following output.

```
Opening file test.txt...

hoping
hope+Prog

realizer
realize+Agentive

realizing
realize+Prog
Closing file test.txt...
xfst[1]:
```

Your transducer MUST rule out:

hope + ing = hopeing
realize + ed = realizeed
flee + ing = fleing

And

```
xfst[2]: apply up hoping
```

should give only one answer.

3 HMM Problems

Go here for the HMM tagging problems.

4 A Naive Bayes classifier problem (prep)

Your Naive Bayes problem is deferred until the final. For that problem it will be helpful to use a **Jupyter Notebook**, as well as NLTK. So your only task on the midterm is to make sure make sure you can use **Jupyter Notebook**, the NLTK movie review corpus, and NLTK.

1. Download

   http://gawron.sdsu.edu/compling/course_core/assignments/midterm_2017/text_classification_naive_bayes.ipynb,

   which is a Jupyter Notebook file. This provides you with some Python code and explanations of how to use it.

2. You can view the notebook by downloading it to your computer, and opening up a terminal window, and connecting to the directory you downloaded the notebook file to, and **in the terminal window**, not in Python, executing the following:

   ```
   jupyter notebook
   ```

   This brings up your browser with a directory window. Clicking on the file will let you view it.

3. If you want to, you can do more than just view the code examples in a Jupyter notebook. You can run them in your browser while viewing the notebook. You can learn more about to use a Jupyter notebook **here**.

4. If you don't have Jupyter Notebook, look **here** (anyone with a full Enthought Canopy or Anaconda installation does have it)

5. It is also highly recommended to install **IPython**, if you don't already have it (anyone with a full Enthought Canopy or Anaconda installation does have it).

Next, read through Part One of the notebook, which contains some code for classifying movie reviews and evaluating the classifier. Make sure that code works (it requires NLTK and the NLTK Movie corpus).

Finally, Part Two of the notebook describes your **task**. You will receive that on your final.

5 A final Thought

Good luck!