Machine Learning, Lab 1
Due: Monday, February 3 at 7pm

LAB 1: MARKOV TEXT GENERATION

ASSIGNMENT AND EXPLANATION

The purpose of this program is to (1) give students practice using Python and command line tools we will continue to use throughout the semester and (2) give students practice working with conditional probabilities while also providing some time for students to catch up.

Write the following Python program. Treat labs like homework assignments. This lab is worth 10 points. You may find the code in provided example code to be useful.

1. Draw the probabilistic finite state machine represented by the following probabilities with either Mermaid, which is integrated into Mark Text (open source), Typora, some VS Code plugins, and several other Markdown editors; or GraphViz's `dot` program. See the provided example code for a GraphViz in example. (Type `make` to build the example graph.)

- The first letter is always "I" (“eye”, not "el").
- The subsequent letters will be generated with the following probabilities:
- You may use spaces instead of underscores; it doesn't matter.
- P(A | _) = 0.5
- P(L | _) = 0.5
- P(M | A) = 0.4
- P(L | A) = 0.6
- P(_ | M) = 0.8
- P(! | M) = 0.2
- P(I | L) = 1
- P(_ | I) = 0.2
- P(N | I) = 0.25
- P(V | I) = 0.55
- P(E | V) = 1
- P(E | N) = 1
- P(! | E) = 1
- P(_ | !) = 0.7
- P(I | !) = 0.2
- P(! | !) = 0.1

2. Write a Python program that outputs the result of this Markov model. The program will output 100 letters per line, on 10 lines. It will output *nothing else*. Do not prompt the user or add any other output or you will lose points. You can access `stdin` and `stdout` streams directly by importing from `sys`, which will allow you to print without newlines.

```
from sys import stdin
from sys import stdout
stdout.write("Hello\n")
```

Alternatively, you can use `print` with an extra optional argument.

```
print("a string", end = '')
```

This program must be runnable **from the command line** with the `python3` command.

Hint:

One possible way of doing this is the following:

```
p = dict()
p["A"] = list()
transitions = ["B", "C"]
probs = [0.5, 0.5]
p["A"].append(transitions)
p["A"].append(probs)
print(p["A"])
print(p["A"][0])
print(p["A"][1])
```

Another is by using a defaultdict, which simplifies the code versus a `dict`:

```
bigrams = defaultdict(lambda: defaultdict(int)) #creates a 2D defaultdict, where the default values is an integer, 0.
bigrams["computer"]["science"] = 5 # example of setting value
print(bigrams["hello"]["world"] # does not exist at query time, so these keys are created with a default value of 0.
```

Yet another solution uses only if-statements, but it may be tedious.

3. Create a simple Makefile that runs your program after typing `make` at the command line. A Makefile is a text file that contains scripts for building and running code, making the process automatic and repeatable. It is always named `Makefile` (case sensitive). To run a Makefile, navigate into the directory containing the Makefile and type `make`.

   Each Makefile has one or more `targets`. A `target` is a section of the Makefile that contains a list of commands to be run in order. Consider the following Makefile:

```
all: target1 target2 plot.jpg
    python3 my_program.py

target1:
    python3 prog1.py
    python3 test1.py

target2: target1
    python3 prog2.py

plot.jpg: generate_images.py:
    python3 generage_images
```

This Makefile has four targets: `target1`, `target2`, `plot.jpg`, and `all`. A colon follows the target name.

If the user types `make target1`, the Makefile will execute `python3 prog1.py` and then `python3 test1.py`. All of the commands run under a target are preceded by a single `[TAB]`. If you just type `make` with no target specified, the first target will be run by default (usually `all`).

More advanced: While not necessary for this lab, note that `target2` and `all` have *dependencies* that follow the colon. The target `target2` depends on `target1`. The `make` program will therefore run `target1` before `target2.`

Targets and dependencies can also be filenames, as in the `plot.jpg` target. This is useful when your program generates output In this case, `make` will check whether the file (in this case, `generate_images.py`) has changed since you last ran the target `plot.jpg`. It does this by checking whether to timestamp of the target is later than that of the dependency. So, if `generated_images.py` is newer than `plot.jpg`, this target will be run. Otherwise, this target will not be run. In that case, running `make all` will also skip the `plot.jpg` target, since there is nothing new to be done.

SUBMISSION

5. Upload two files: first, a PDF file with your report (written in Markdown or LaTeX), which should contain your output in a fenced code block and your finite state diagram.

   Also upload your Python file and Makefile in a `tar.gz` file called `lastname_firstname.tar.gz`, which contains your actual last name and first name. Instructions for creating a compressed gzip file can be found here.

Points will be allocated based on correctness of the program and whether you followed the directions. I do not grade assignments that do not follow the instructions.

Try re-downloading your files to ensure that they are the correct ones. I will grade whatever is there.