Homework 10

Collin Stewart
https://github.com/collings512/BIOS512_Collin_Stewart

Course Notes

Language Models: https://github.com/rjenki/BIOS512/tree/main/lecture17
Unix: https://github.com/rjenki/BIOS512/tree/main/lecture18
Docker: https://github.com/rjenki/BIOS512/tree/main/lecture19

In [ ]:
```
install.packages("httr")
install.packages("tokenizers")
install.packages("dplyr")
install.packages("tidyverse")
install.packages("stringr")

library(httr)
library(tokenizers)
library(dplyr)
library(tidyverse)
library(stringr)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependency ‘SnowballC’


Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)


Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union


── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ forcats   1.0.1     ✓ readr     2.1.6
✓ ggplot2   4.0.1     ✓ stringr   1.6.0
✓ lubridate 1.9.4     ✓ tibble    3.3.0
✓ purrr     1.2.0     ✓ tidyr     1.3.1
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

Question 1

Make a language model that uses ngrams and allows the user to specify start words, but uses a random start if one is not specified.

a) Make a function to tokenize the text.

In [ ]:
```r
tokenize_text <- function(text) {
  tokenizers::tokenize_words(text, lowercase=TRUE, strip_punct=TRUE)[[1]]
}
```

b) Make a function generate keys for ngrams.

In [ ]:
```r
key_from <- function(ngram, sep = "\x1f") {
  paste(ngram, collapse=sep)
}
```

c) Make a function to build an ngram table.

In [ ]:
```r
build_ngram_table <- function(tokens, n, sep = "\x1f") {
  if (length(tokens) < n) return(new.env(parent = emptyenv()))
  tbl <- new.env(parent = emptyenv())
  for (i in seq_len(length(tokens) - n + 1L)) {
    ngram <- tokens[i:(i + n - 2L)]
    next_word <- tokens[i + n - 1L]
    key <- paste(ngram, collapse = sep)
    counts <- if(!is.null(tbl[[key]])) tbl[[key]] else integer(0)
    if(next_word %in% names(counts)) {
      counts[[next_word]] <- counts[[next_word]] + 1L
    } else {
        counts[[next_word]] <- 1L
    }
    tbl[[key]] <- counts
  }
  tbl
}
```

d) Function to digest the text.

In [ ]:
```r
digest_text <- function(text, n) {
  tokens <- tokenize_text(text)
  build_ngram_table(tokens, n)
}
```

e) Function to digest the url.

In [ ]:
```r
digest_url <- function(url, n) {
  res <- httr::GET(url)
  txt <- httr::content(res, as = "text", encoding = "UTF-8")
  digest_text(txt,n)
}
```

f) Function that gives random start.

In [ ]:
```r
random_start <- function(tbl, sep = "\x1f") {
  keys <- ls(envir = tbl, all.names=TRUE)
  if(length(keys)==0) stop("No n-grams available. Digest text first.")
  picked <- sample(keys, 1)
  strsplit(picked, sep, fixed=TRUE)[[1]]
}
```

g) Function to predict the next word.

In [ ]:
```r
predict_next_word <- function(tbl, ngram, sep="\x1f") {
  key <- paste(ngram, collapse = sep)
  counts <- if(!is.null(tbl[[key]])) tbl[[key]] else integer(0)
  if (length(counts) == 0) return(NA_character_)
  sample(names(counts), size=1, prob=as.numeric(counts))
}
```

h) Function that puts everything together. Specify that if the user does not give a start word, then the random start will be used.

In [ ]:
```r
make_ngram_generator <- function(tbl, n, sep = "\x1f") {
  force(tbl); n <- as.integer(n); force(sep)
  function(start_words = NULL, length = 10L) {
    if ((is.null(start_words)) || length(start_words) !=n - 1L) {
      start_words <- random_start(tbl, sep=sep)
    }
    word_sequence <- start_words
    for (i in seq_len(max(0L, length - length(start_words)))) {
      ngram <- tail(word_sequence, n - 1L)
      next_word <- predict_next_word(tbl, ngram, sep=sep)
      if(is.na(next_word)) break
      word_sequence <- c(word_sequence, next_word)
    }
    paste(word_sequence, collapse= " ")
  }
}
```

Question 2

For this question, set `seed=2025`.

a) Test your model using a text file of Grimm's Fairy Tails

i) Using n=3, with the start word(s) "the king", with length=15.

ii) Using n=3, with no start word, with length=15.

In [ ]:
```r
set.seed(2025)

urlques2a <- "https://www.gutenberg.org/cache/epub/2591/pg2591.txt"
tblques2a <- digest_url(urlques2a, n=3)
genques2a <- make_ngram_generator(tblques2a, n=3)


print(genques2a(start_words = c("the", "king"),length=15))

set.seed(2025)
print(genques2a(length=15))
```

```
[1] "the king has forbidden me to marry another husband am not i shall ride upon"
[1] "spread the jam over it spread its wings and crying here comes our hobblety jib"
```

b) Test your model using a text file of Ancient Armour and Weapons in Europe

i) Using n=3, with the start word(s) "the king", with length=15.

ii) Using n=3, with no start word, with length=15.

```
In [ ]: set.seed(2025)

urlques2b <- "https://www.gutenberg.org/cache/epub/46342/pg46342.txt"
tblques2b <- digest_url(urlques2b, n=3)
genques2b <- make_ngram_generator(tblques2b, n=3)


print(genques2b(start_words = c("the", "king"),length=15))

set.seed(2025)
print(genques2b(length=15))
```

```
[1] "the king he added to the entire exclusion of the swords were made prisoners th
e"
[1] "lamentation de lemburn came forth completely armed after the fashion of this ma
y be seen"
```

c) Explain in 1-2 sentences the difference in content generated from each source.

Since the input/previous/reference source text for each example is different (Grimm's Fairy Tales vs. Anicent Armour and Weapons in Europe), the predicted text will be different. This is the case for having both no text (random start word determined by the seed), or having the two words "the king" preceeding the prediction.

Question 3

a) What is a language learning model?

b) Imagine the internet goes down and you can't run to your favorite language model for help. How do you run one locally?

a) A language learning model is a machine learning model which predicts the probability of a sequence of words to understand and generate human language. Essentially, it is a probablity distribution over words, given some input of text/words as "previous words", it will predicting the next word.

b) To run a language model locally, you must install the model (example: OLLAMA) to your device, run the model API server, use functions for POST wrapper, chat completion, and factory, then run it.

Question 4

Explain what the following vocab words mean in the context of typing `mkdir project` into the command line. If the term doesn't apply to this command, give the definition and/or an example.

Term	Meaning
Shell	The shell is the program that receives the bytes of 'mkdir project' and interprets them as a command, allowing us to interact with them.
Terminal emulator	The host to the shell, it's the place that the shell sits and runs from. It displays the bytes from the shell in different colors, images, animations, etc.
Process	Something running on the computer. In this case, the mkdir program is the process that runs, executes, and ends.
Signal	When we type mkdir, we send a signal to the process that runs the mkdir program. Something we send to a process to tell it to do something.
Standard input	The component of our process that is able to read the characters in "mkdir project" and understand what it means.
Standard output	The text that is written as an output of the process that executes our "mkdir project" input. Sometimes, there is no output, it just runs.
Command line argument	In this case, "project" is the command line argument. mkdir is the process, and "project" is the argument that tells it what directory to create
The environment	Everything that the process (mkdir) can see when it is running, including variables and settings.

Question 5

Consider the following command `find . -iname "*.R" | xargs grep read_csv`.

a) What are the programs?

b) Explain what this command is doing, part by part.

a) The programs in the command are find, xargs, and grep

b)find is a program that searches for files, and . lets the find program know to look in the present working directory.

xargs is used to basically pipe the output of the first program (all of the R files that match the find program) and puts it into the command line of the second command, rather than the standard input. We're passing not just the names of each .R file to "grep", but the contents of the files themselves to grep.

grep allows us to search for things within files themselves. We use grep read_csv to search for matches of the read_csv within each .R file that was selected from the working directory in the initial find program.

Question 6

Install Docker on your machine. See here for instructions.

a) Show the response when you run `docker run hello-world`.

b) Access Rstudio through a Docker container. Set your password and make sure your files show up on the Rstudio server. Type the command and the output you get below.

c) How do you log in to the RStudio server?

a) After running docker run hello-world, this was the response: Unable to find image 'hello-world:latest' locally latest: Pulling from library/hello-world 17eec7bbc9d7: Pull complete Digest: sha256:f7931603f70e13dbd844253370742c4fc4202d290c80442b2e68706d8f33ce26 Status: Downloaded newer image for hello-world:latest

Hello from Docker! This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with: $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID: https://hub.docker.com/

For more examples and ideas, visit: https://docs.docker.com/get-started/

b) Command:
docker run -d -p 8787:8787 -e bios512a10o90210=mypassword rocker/rstudio

Output:
Unable to find image 'rocker/rstudio:latest' locally latest: Pulling from rocker/rstudio 3665120d345d: Pull complete 3c7cdccc4be7: Pull complete 664fb1818bbb: Pull complete 62f215ca34c6: Pull complete e4b9e87bb831: Pull complete 5d246ec925db: Pull complete

2c9ba66d5dbe: Pull complete 39038e16d1ba: Pull complete 191985778909: Pull complete d923cf803a12: Pull complete 2a63ed8b2250: Pull complete 4b3ffd8ccb52: Pull complete 890065c4c99d: Pull complete 999e4b8f7ed8: Pull complete 9c1a4a0706b7: Pull complete b71e78fefbbb: Pull complete 971ba7cf0d8a: Pull complete 08e74fd5985d: Pull complete
Digest: sha256:9f85211a666fb426081a6f5a01f9f9f51655262258419fa21e0ce38a5afc78d8
Status: Downloaded newer image for rocker/rstudio:latest
781ee025af1a3cb5682b5e36af44cd006c5329174454debb08844d323daa3042

c) To login to the Rstudio server, you would simply type the following into your web browser:
http://localhost:8787

The username will be rstudio, and the password will be the one that I assigned in the previous step.