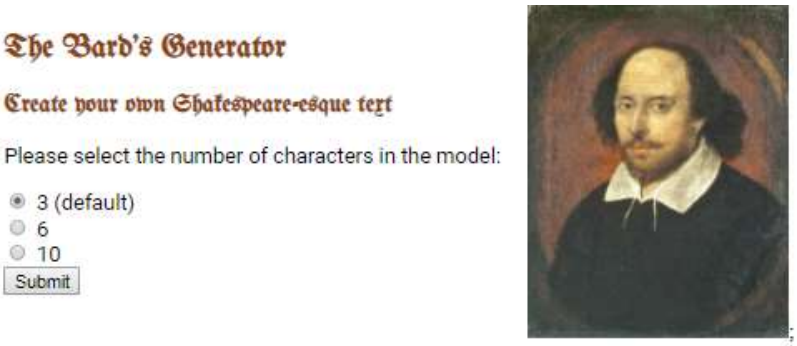
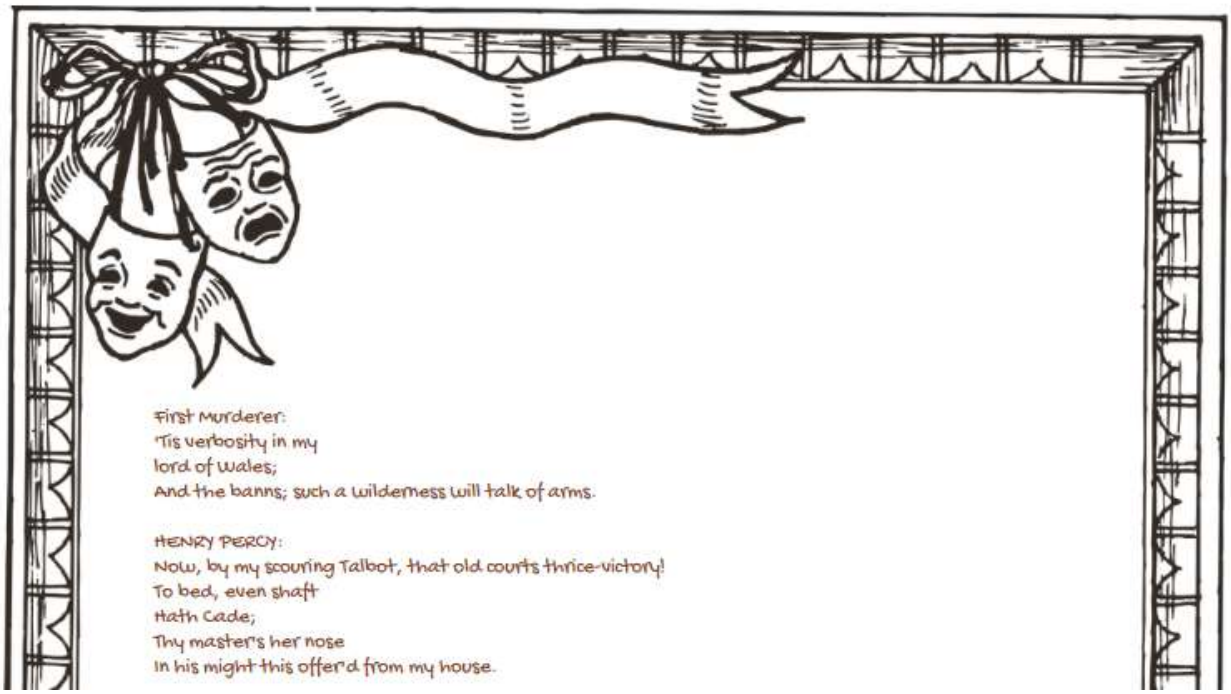
Project 1 – Shakespeare text generator

(individually or teams of 2)





Read the entire specification before starting!

Introduction

Artificial intelligence, machine learning and neural networks are all the rage! You may have heard about fake text generators that are too dangerous to release in the public. There are different deep learning mechanisms that can be used to generate new works, but most are based on using a lot of existing data to predict what could be next in a sequence. Some excellent results are found using recurrent neural networks to predict next words or characters in a sequence (check out this short film!). But in this project we are going to generate Shakespeare-sounding text using a much simpler "character-level language model". This project is largely inspired by this article.

A character-level language model is basically a probability model that predicts the sequences of characters in a given language. As an example, let's say you saw a sequence of 5 characters: `ello`.

What do you think would be the next character? It may be a space, or a punctuation symbol (if the word was `hello`), or it could be a `w` if the word was `yellow`. The more characters we have in the sequence, the easier it is for us to predict what could be a reasonable next character.

A recurrent neural network can potentially work with and remember a very large number of characters in a sequence; this requires a huge amount of processing and some advanced math to understand what is going on (I'm not sure I'm there yet!). We are going to use a limited number of characters, n, and try to guess the $n+1^{th}$ character using simpler probability model.

Let's say you have a lot of Shakespeare's texts. And you want to know the probability of a given character being the next character after a sequence of n characters. Let's call this the probability of character c, given that sequence s of n characters just occurred, or $P(c|s)$. The most intuitive way is to find all the occurrences of s in Shakespeare's text, and count the number of times the different characters appear, dividing by the total number of occurrences of s. This is called a maximum likelihood estimate. If a given character never appears after a sequence, it will have a probability of 0.

Once you have the probabilities, you will generate the next character in a sequence. But you can't only use the character with the maximum likelihood of appearing, or else your text will always be the same given that same starting sequence. Instead you will choose a random character based on the distribution. Note that we will force the first characters to always be the first characters in the original Shakespeare text.

There are three parts to this project.

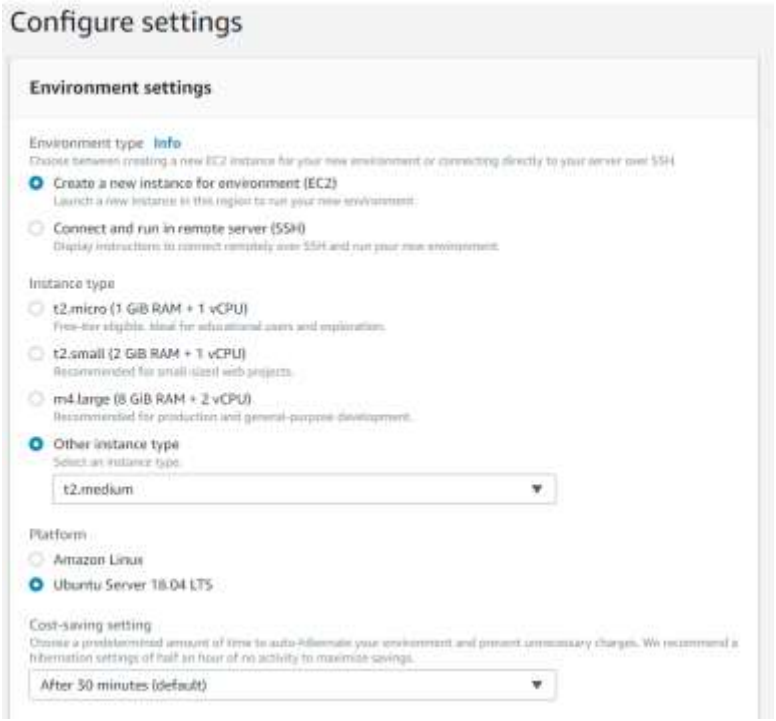
- first, create your Cloud9 development environment
- then you will train your model and update a local development Redis data store; this is written in a console application. Redis is an in-memory data store that is much quicker than a relational database for certain types of data:
    - simple data with a key-value model, where the keys are strings
    - large amounts of data which will take long to write or query from a relational database on a disk
    - data which does not need a schema/table structure or relationships (Redis is a NoSQL data store)
- next, you will write a php script with a form that asks the user for the number of characters (3, 6, or 10 to use in the probability model); that you will use this to query the Redis data store and generate the text.

Use best coding practices: modularize in functions or classes, use the data access object design pattern for separation of concerns, validate user input.

Each of the partners should work in a new Cloud9 environment (don't use the same one we use for labs). Use standard git practices to make sure you are working on up-to-date code.

Part 1 – Start a new Cloud9 environment

The data store that we end up creating is quite big so the memory requirements are also big. Luckily, we are using a cloud-based system with free credits from our teacher! When you get to the Cloud9 environments dashboard, choose other instance type, then t2.medium.



You can use either Amazon Linux or Ubuntu, the main differences from our perspective being that Amazon Linux uses `yum` and Ubuntu uses `apt` as package managers. The instructions below assume Ubuntu on Cloud9.

Remember to update the software on the new Cloud9 environment:

```
sudo apt -y update

sudo ln -s /bin/nano /usr/bin
```

Your instructor has created a new repository (project in GitLab terminology) for the assignment. Set your git configuration and clone the repo created for you:

```
rm README.md #to avoid merge conflict

git config --global user.name "Your Name"

git config --global user.email you@dawsoncollege.qc.ca

git clone youGitLabURL/ass2.git

mv ass2/.* .

rm -r ass2

echo .c9/ > .gitignore
```

Edit the README.md file.

And finally, you may want to go into preferences (the gear icon to the right), scroll down, and stop your environment after an hour (so that you don't have to keep logging back in).

Part 2 – Training the model

You will write a script, split into functions and/or classes, which is invoked from the console. It will iteratively read the file and populate an array and then write to a local Redis data store.

We need to install Redis on Cloud9. First we will install Composer, the PHP package manager:

```
curl -sS https://getcomposer.org/installer | php

sudo mv composer.phar /usr/local/bin/composer
```

Next, install Predis, which is the PHP Redis client:

```
composer require predis/predis
```

When we use Composer, it installs in the vendor folder. It is better practice to not track the vendor files in version control, and instead to install based on the composer.json file on any new computer (using the command `composer install` which will create the vendor folder for you). So add to your `.gitignore` (you can use nano, vim or the Cloud9 IDE if you click on the settings – show hidden files):

```
vendor/
```

Finally, install Redis locally:

```
sudo apt install redis
```

Check that it is listening on port 6379:
```
ss -nlt
```

Optional: Make sure it starts after a reboot
```
sudo systemctl enable redis-server
```

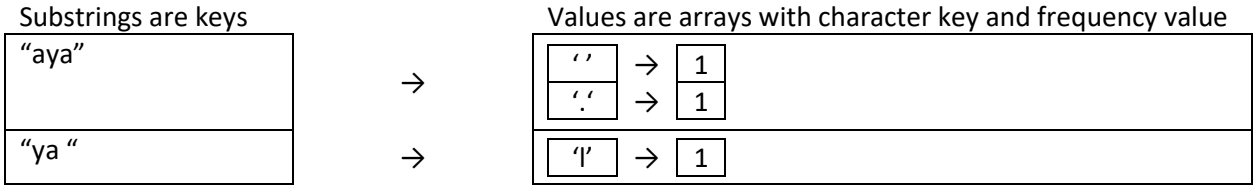
Start coding!

Do NOT write your code in the master branch! Make a staging branch, and from there multiple development branches. Merge features into the staging branch, using Merge Request: the other partner MUST review the code before accepting the MR. Do not merge into master!

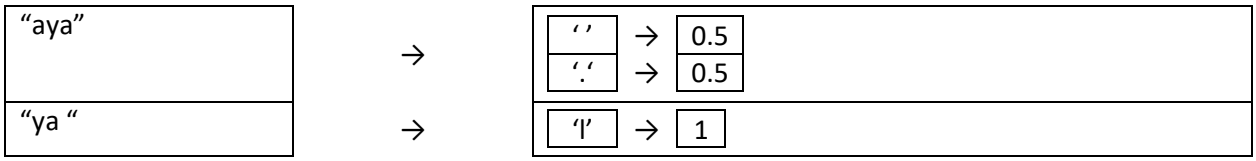
We will be storing key-value pairs in Redis: the key is each substring of text (either 3, 6, or 10 characters long) and the value is a json-encoded associative array, with each character that follows the particular sequence, and its frequency. First populate a PHP array before saving it in Redis.

For example, if the text is: "Jaya loves papaya.", you would have two entries in the table for "aya", one being 'aya' and '.' with occurrence 1, and the other being 'aya' and ' ' with occurrence 1. Each of these characters have a probability of 1/(1+1) = 0.5.

The associative array of arrays will look like this (partially), for example:



Once you reach the end of the text, iterate through the array, and change the frequency values to percentages, to reflect the probability:



You will read the file and populate an associative array of arrays:

- read the Shakespeare input file (shakespeare_input.txt, from Andrej Karpathy)
- iterate through the text string, starting with substrings of length 3, and use as the key in the sequences associative array
    - add a new key if it is the first time the sequence is found
    - the value associated with the substring is an array with the next character as key and the frequency as value
- repeat the process for substrings of size 6 and 10.

In order to save into the Redis data store using Predis, create a connection to the Redis server:

```
require "predis/autoload.php";
Predis\Autoloader::register();

try {
  $
$redis = new Predis\Client( [
    "scheme" => "tcp",
    "host" => "localhost",
    "port" => 6379] );

     //Your code
} catch (Exception $e){
    //Note: bad practice to catch general exception, but you are exiting
    error_log('error with Redis '. $e->getMessage() );
    exit;
}
```

I recommend that you use a separate variable file to hold the host and other Redis connection variables to simplify the changes needed when you deploy in production.

Look at the Predis and Redis documentation to get an idea of how you will save the key-value pairs saved. Unlike relational databases, there are no table schema or relationships. Methods that will be helpful:

$redis -> set ($key, $value) where both are strings. Since our values are associative arrays, json_encode them
$redis -> get ($key) to retrieve the associative array of character probability. If there are none, NULL is returned, which you must treat. Remember that you need to json_decode the return value.
$redis -> flushAll() to clear the data store. Do this everytime you run your script that fills the store as you are testing.

You will run from the console in Cloud9: `php yourscriptname.php`

PRO TIP: Test your code iteratively (i.e., as you add more functionality). For example, first test your code that generates an associative array on a short string; make sure that the boundary conditions (start and end of string) are properly dealt with. When you are testing with Redis, and want to see the results, use the `redis-cli` tool. Commands that will be helpful:

```
redis-cli DBSIZE        - get the number of keys
redis-cli GET fir       - gets the value associated with "fir"
```

Remember to frequently commit your code to a GitLab development branch, using best practices.

It will take approx. 10 minutes for the training to be complete. You should have 3.6 million keys in your data store!!! (as an aside, it would have taken a long time to write all these rows into a relational database and it would cost too many credits to write this into the AWS NoSQL database).

Make sure that you modularize your console application with either functions or classes.

Part 3 – Generating Shakespeare-like text

In this part, you will write scripts, split into functions and/or classes, which are invoked from the web (remember to name the home page `index.php`). The form will ask the user to choose between 3, 6 and 10 character models. Based on the choice of number of characters and the generated string will start with "Fir", "First ", and "First Citi". (nb: this is hard coded to start with the same text as the file with all Shakespeare text. You can choose another starting point, but choose something that would be a natural starting point and that appears in Shakespeare's texts).

Based on the starting point and character model, repeatedly use the sequence of the last n (3, 6 or 10) characters from the result string to generate the n+1 character which is appended to the result string. Keep generating 1000 characters, then display the generated text on screen.

How do you create an individual character? We will weight our character selection according to the frequency of characters that followed the particular text sequence.

- Based on the sequence of last n characters (HINT: remember what a negative substring length means?), query Predis to find all the characters that follow, and their probability
- generate a random number between 0 and 1 that represents a probability (HINT: PHP's rand() returns a random integer between 0 and getrandmax(), divide that random integer by the maximum possible integer getrandmax() to get a float between 0 and 1
- find out which character corresponds to the random probability

For example, if the text 'ello' has the following subsequent character frequency

'!'	' '	','	'w'	'r'	'.'
0.2	0.1	0.1	0.4	0.1	0.1

and your random number is 0.55, you generate character 'w'; if your random number is 0.26, you will generate ' '.

What happens if a particular substring is not in Shakespeare's work? It actually is not possible since we are building the strings to only contain characters that have a non-zero probability of being found! Since you need to programmatically deal with this condition though, simply return your favourite letter!

HINT: when you are echo-ing the resulting Shakespearean string, the new line characters won't display since they are not how html displays line breaks. The nl2br function is your friend!

Finally, if you use any images or fonts, please include a reference in the footer. Since this is an academic project, you can use any image; but if you want to eventually include this in a portfolio, I recommend you use royalty-free images (from Pixabay or Unsplash, for example).

Submission

You will submit your code to the GitLab repository created for your team. Note that you should NOT write code to the master branch, so you will work in branches. Please continue to use the protected branch workflow that you have used the past two semesters; make sure you make a merge request to master when you submit.

We will not deploy this project to a production system– there is a huge risk that you will use all your AWS classroom credits since the data set is so huge and AW$ is greedy and we have two more projects plus labs to do. We'll deploy to production in the next project.