CA247 ASSIGNMENT 1

Building a Language Recogniser

Jamie Somers & Daniela Oana

Group 106

Introduction to the System:

Below, the *inputs*, *processes* and *outputs* are discussed in relation to the Language Recogniser system.

Inputs:

For *train.py,* the initial input is the language_input_files.txt file which contains both *english_training.txt* and *french_training.txt*. The system then uses both of those file names to open said files to retrieve the strings to use for training purposes.

For *test.py,* the input reads in the character probability distributions that are calculated and outputted from *train.py*. It will also require a string to be inputted into the console, which the program will use to determine if it is in English or in French.

Processes:

In *train.py*, for loops are utilised to open each of the files to reach the test data in English and French. Then, the for loops iterate through each line of data and appends each character to a list. The next process counts the frequency of each of the characters in both English and French and puts them into a dictionary. After that, the required calculations are carried out to obtain the probability distributions of each character. Finally, the values are transferred into an array and written to a file.

In *test.py*, the first process reads in the values of the probability distributions from the output file from *train.py* and adds them to their respective lists, English and French. Using lists, the probability distribution data is separated until there is a list containing the letters and another list containing the values, again for each language. Once the user enters a string, the final process calculates which language the string is most likely to be based on the overall probability of each of its characters multiplied together. The larger probability decides the language and this is printed to the screen.

Outputs:

For *train.py*, the output is the calculated character probability distributions which are displayed in a text file that the program creates.

For *test.py*, the output displays which language the inputted string is most likely to be, i.e. English or French. It selects the appropriate language by determining the greatest probability.

Data Pre-Processing:

Another fundamental aspect of the system is the data that is used. Before implementing the data, it is necessary to pre-process it to ensure that it is usable. In this instance, the data is strings in both the English and French languages. Each of the three English and three French training files contained simple sentences. The same sentences were used for both English and French, translated into their respective languages.

All punctuation and capitalisation was removed from the data in both languages as to only feature one type (lowercase) of each letter of the alphabet in the distribution. Since the French language contains accents and special characters, these had to be removed and replaced with the standardised “plain” variant of the letter. Once the program was ensured to be correct, the simple sentences were replaced with larger chunks of text to increase overall accuracy. The first four thousand characters from the English and French versions of the book “The Adventure of the Engineer’s Thumb” by Arthur Conan Doyle were used.

Generated Plots:

The following plots have been created using *matplotlib*, in relation to the data from both the *train.py* and *test.py* programs. Firstly, bar charts based on data from *train.py*.

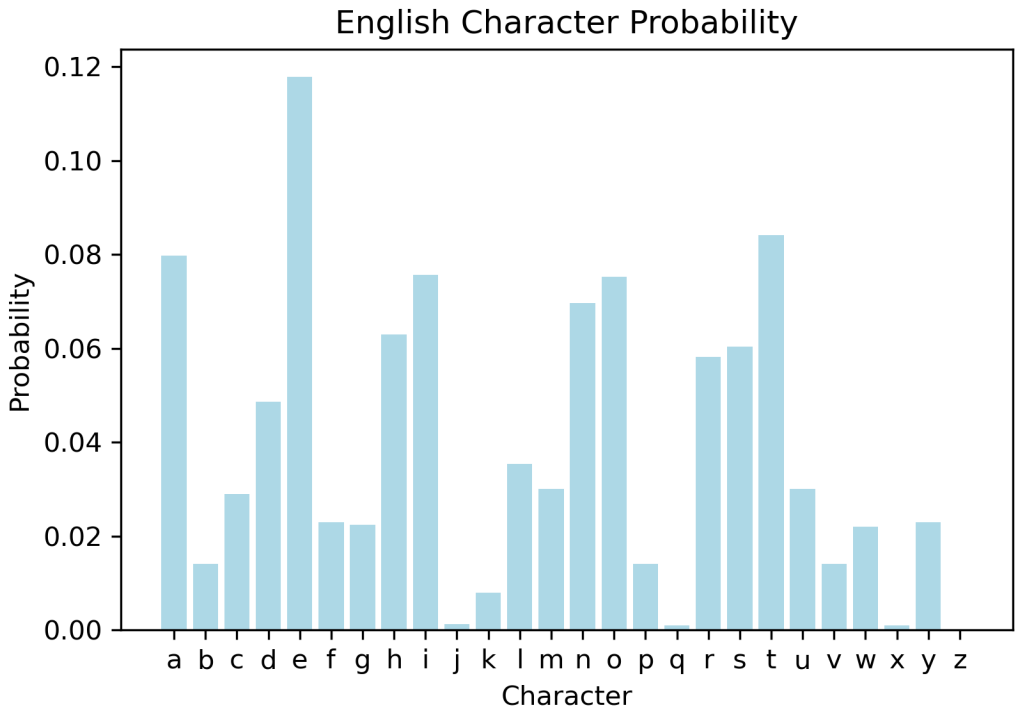

Figure 1: Bar chart showing English character probabilities

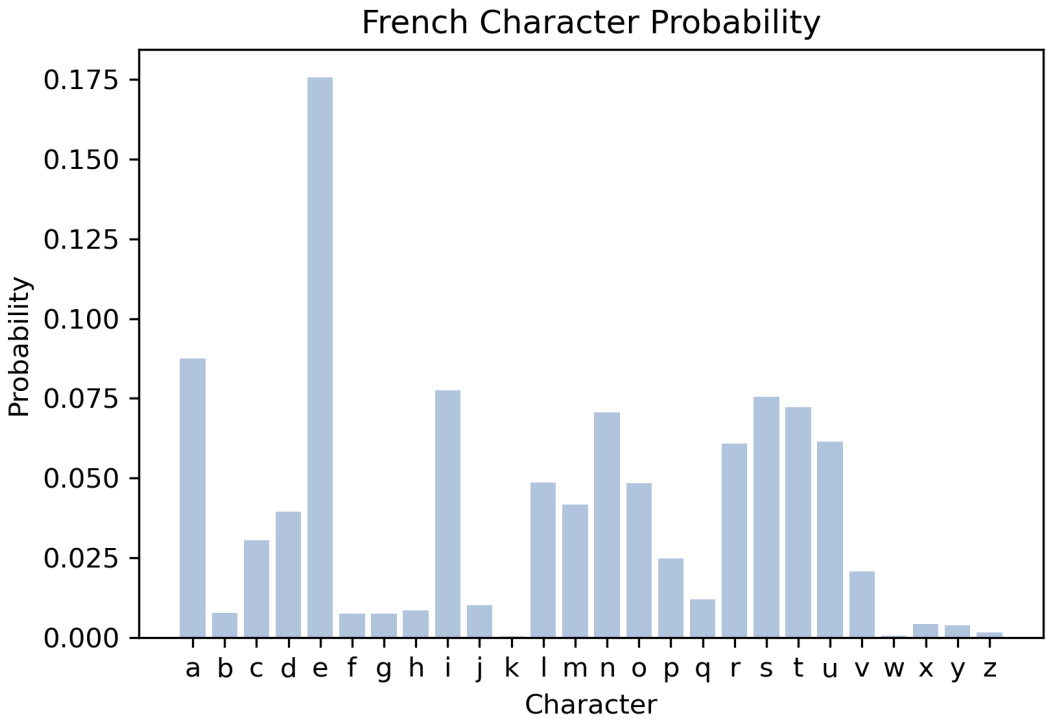

Figure 2: Bar chart showing French character probabilities

Secondly, a pie chart based on data from *test.py*.

16 English words and 16 French words were entered into the program to test its accuracy. The expected result would be 50% of the words would return as English and the other 50% would return as French.

Table 1: English and French words used to test accuracy

English	French
physics	la physique
engineering	ingenierie
dog	**chienne**
potato	pomme de terre
laboratory	**laboratoire**
icecream	la creme glacee
cup	tasse
cylinder	**cylindre**
computer	lordinateur
mathematics	mathematiques
people	**gens**
time	temps
day	jour
man	**homme**
woman	femme
bee	abeille

The words in **bold** were recognised incorrectly by the program.

The pie chart below in **Figure 3** represents the actual result.

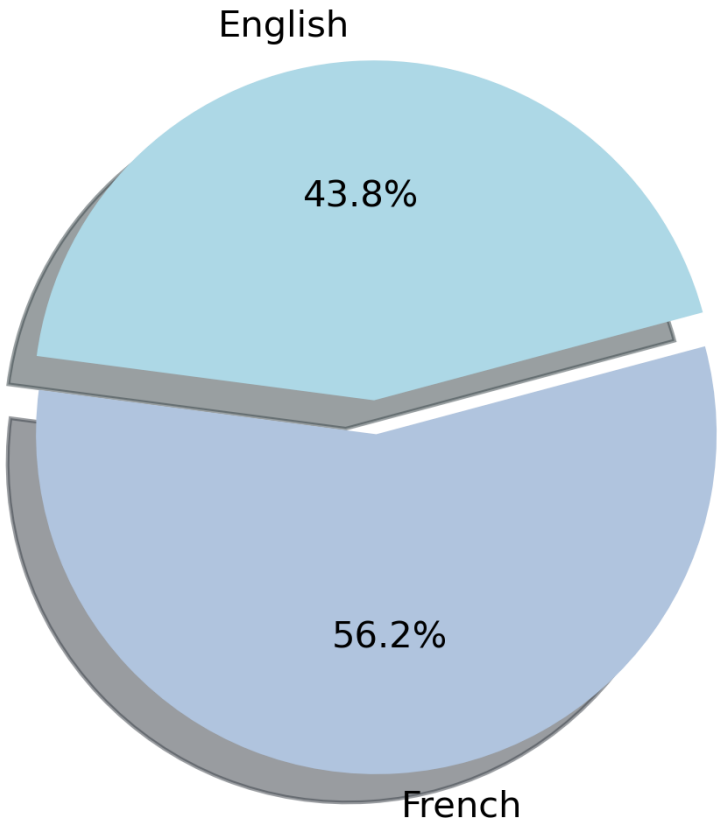

Figure 3: Pie chart representing French bias

Examining the pie chart, there is a slight French bias. This means that the program recognised some of the English words incorrectly and outputted French instead.

The letter ‘e’ has a much higher probability in French compared to English. It is also the most common letter for both languages. Five out of seven of the English words recognised as French by the program contained the letter ‘e’. Hence, this may be the reason as to why the program chose incorrectly.

Experimenting with the System:

To further test the functionality of the code, an experiment was carried out. This experiment investigated the relationship between the length of test strings and the accuracy of the language recognition. A sample size of ten words per length for varying lengths was used from the range of two characters to six characters. The expected result was that the shorter the word, the less accurate the program was at recognising the language.

Table 2: Words of varying lengths used as test data

2	3	4	5
ay	les	dans	adieu
eh	des	pour	pomme
ey	que	elle	avoir
al	une	mais	livre
le	est	vous	porte
ba	cat	rock	would
ad	him	**nest**	write
is	off	chop	robin
or	**our**	**tent**	**exile**
us	**red**	wink	where

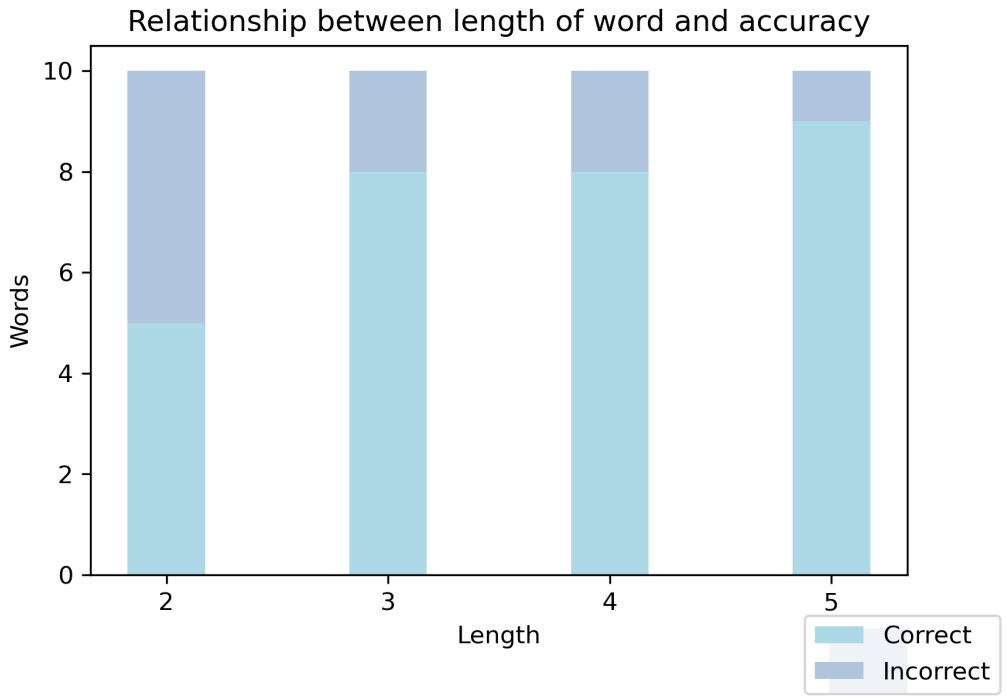

Figure 4: Stacked bar chart relating length and accuracy

Conclusions:
After carrying out the experiment, the expected result proved to be correct. The shorter the word, the less accurate the program was in recognising the correct language. Therefore the longer the word, the more accurate the program was, reaching 90% accuracy with five character length words.
Again, the French bias is noticeable here as there are more incorrect English words than French words.

Limitations of the System:

Of course with any system, there will always be some limitations to the code. The following limitations were discovered with the code for this program.

- The shorter the length of the string, the greater the difficulty the program has in accurately interpreting the language. Short words tend to contain similar characters, particularly vowels. If the string is longer, there is a higher chance of more unique characters appearing. For example, in English ‘w’ and ‘y’ are more unique whilst in French ‘j’ and ‘q’ are more unique.

- Some words are also exactly the same in both languages, whether they mean the same thing or not. For example, ‘chat’ in English and ‘chat’ meaning cat in French. The program could not determine that ‘chat’ is both an English and French word.

- The values of the probability distributions are rounded to four decimal places to keep them tidy and to avoid long trailing numbers. However, if a large number of characters was used as test data and a specific character appeared very little, its probability could have a smaller order of magnitude than 4 decimal places and assume it is 0.

Further Work:

If this project was continued beyond the allotted time and given requirements, further amendments and expansions to the code could be made.

Firstly, working on bypassing the limitations of the system would be a priority. For example, to combat the difficulty in determining short strings, more test data could be used to create a greater spread of probabilities.

Certain words could be hardcoded into the program such as ‘chat’ where the output would display “inconclusive” as this word is both English and French.

Furthermore, extra languages could be added to the system such as German and Spanish to expand on the program. This of course would require more training data and more files to be read into *train.py*.

Addendum:

Who did what:

Jamie

- Programmed reading in and opening multiple files **(train.py)**
- Sorted characters and probabilities into lists **(train.py)**
- Read in Probability distributions **(test.py)**
- Created input string from user **(test.py)**
- Programmed language determination **(test.py)**
- Created French probability bar chart **(Figure 2)** and stacked bar chart **(Figure 4)**
- Wrote **inputs** and **processes** in **Introduction to the System**
- Carried out test for accuracy **(Table 1)**
- Wrote **Limitations of the System**

Daniela

- Determined the frequency and sum of each character **(train.py)**
- Programmed writing probabilities and characters to text file **(train.py)**
- Sorted English / French letters and values into lists **(test.py)**
- Programmed the probability equation **(test.py)**
- Output formatting and printing **(test.py)**
- Created English probability bar chart **(Figure 1)** and pie chart **(Figure 3)**
- Wrote **outputs** and **data processing** in **Introduction to the System**
- Carried out experiment test for accuracy **(Table 2)**
- Wrote **Further Work**

Problems encountered/overcome:

- When creating the train.py code for opening multiple files, there were many variables to keep track of, this resulted in errors such as files opening in the incorrect order. e.g *french_training01.txt* opening before *english_training03.txt*. This was solved by ensuring our code was iterating in the correct order and file orders hadn't been confused.

- While trying to calculate the probability of each character appearing in *train.py*, the probability calculation was carried out within the same loop as the sum calculation. This resulted in the probability getting linearly smaller proportionally to the sum variable getting bigger. This was solved by calculating the sum first, and then calculating the probability in a separate loop.

- When importing the probabilities to *test.py* from *Probability_Distrubutions.txt*, there was a problem where our formatting caused the string 'English:' and 'French:' to be imported into a list instead of the first probability. We solved this by skipping one line and taking the proceeding 26 lines.

What could be done differently:

- Code could have been optimised to better take advantage of the loops available within python, some sections of the code relating to French and English are identical copies with only one or two variables changed instead of one piece of coding with iterating variables.

- Lists were used multiple times one after another in test.py, more time to optimise the code could have reduced the number of lists needed, or at the very least the number of lines of code to make this work.

- Due to the level of familiarity, occasionally lists were used in circumstances where dictionaries may have been more appropriate.

- It's possible that implementing the code within a function would have been more appropriate, this way the entire function could be copy and pasted into future files where a language trainer is required.