SOLVING ODD-ONE-OUT IQ TEST PROBLEMS WITH A COMPUTER MODEL

FIT3036 COMPUTER SCIENCE PROJECT
SEMESTER ONE

DAVID NGUYEN
DNGU71@STUDENT.MONASH.EDU.AU
26882434

MONASH UNIVERSITY
AUSTRALIA

ABSTRACT

WITH EACH TECHNOLOGICAL BREAKTHROUGH, NEW TECHNIQUES ARE TESTED AND ADAPTED TO SOLVE OLD QUESTIONS. THE RISE IN POPULARITY OF MACHINE LEARNING TECHNIQUES LEADS INDIVIDUALS, AND TEAMS TO TACKLE PROCESSES ONLY HUMANS HAVE EVER COMPLETED BEFORE. AFFECTS OF THIS CAN BE SEEN AS MACHINES ARE BEGINNING TO BEAT HUMANS AT GAMES WE CONSIDERED TO BE MASTERS OF. WE WILL VERY SOON SEE THESES TECHNIQUES DIRECTED AT SOLVING IQ TEST PROBLEMS. PRIOR TO THAT HAPPENING, THIS PROJECT WILL HOPEFULLY SHOW THAT SOMETIMES JUST HAVING ENOUGH INFORMATION IS BETTER THAN PURELY RELYING ON MACHINE LEARNING TECHNIQUES. THIS PROJECT'S COMPUTER MODEL UTILIZES A VAST AMOUNT OF INFORMATION TO FIND LINKS BETWEEN ITEMS TO SOLVE ODD ONE OUT QUESTIONS. THIS MODEL WAS DESIGNED AND DEVELOPED TO ALWAYS HAVE UP TO DATE INFORMATION BY OBTAINING ITS DATA FROM ONLINE SERVICES. AFTER ANALYSING THE RESULTS OF THIS COMPUTER PROGRAM, IT WAS DISCOVERED THAT WITH ENOUGH INFORMATION, SOLVING ODD ONE OUT QUESTIONS BECOMES EXCEEDINGLY EASY AND ACCURATE.

Keywords ODD ONE OUT, ODD MAN OUT, CATEGORIES SOLVING PROBLEMS, COMPUTER MODEL

28TH MAY 2018
APPROXIMATELY 4,650 WORDS

Contents

1 Introduction

Computers have been trying to solve IQ test problems for decades. Since then, there are now programs that have been developed that can beat humans at certain IQ tests in the early 21st century [1]. However, a computer solving ‘odd-one-out’ IQ test questions perfectly has still been proved difficult in these recent years.

Odd-one-out problems typically involve providing the user with a set of 4 to 5 items, where one item does not belong in that set. There are many different types of odd-one-out questions. Such as selecting the odd shape out, odd image out, odd word out, odd object out, or selecting the odd number out.

Recently in the 21st century, with rapid technological advancements, there has been a renewed interest in researching methods and techniques for computer programs to complete IQ test questions. New research into machine learning has allowed machines to answer certain types of odd-one-out questions [2]. There have also been physical robots researched and developed solve odd-one-out questions as well [3].

This project will be aiming to solve selecting the odd word out, also known as ‘odd man out’ type of questions. Previously there has been work done on IQ questions by utilizing search engines [4]. Attempting odd man out questions using the same search engine technique proved difficult, due to the results found being irrelevant to the actual answer.

Instead of using search engines, this program will utilize a large data set of categories describing a large range of items. An algorithm will then search through the data and find matching categories to determine the correct odd man out answer.

1.1 Objectives

The main objective of this project is to design and develop a computer program that is capable of solving odd man out questions. The secondary objective tune the program to obtain a high solving percentage of odd man out questions.

1.2 Requirements

As previously shown in the project specification [5], the project requirements are shown below, albeit with some updates.

1.2.1 Functional Requirements

- Ability to receive initial inputs from the user

- Ability to retrieve data from Wikipedia using their MediaWiki's query API (application programming interface)
- Ability to interpret data to determine which input has the highest probability of being the odd one out
- Ability to display results and errors to the user

1.2.2 Non-Functional Requirements

- The application must be able to handle 3 to many inputs
- The application must work on all platforms that are capable of running Python version 3.6 and above

1.3 Constraints

The project shall be given the following restraints due to time constraints from other units:

- Only handle items that have a page on Wikipedia
- A maximum of 4 weeks to build and test the program
- Don't handle date or pronunciation type questions

2 Background

2.1 Literature Review

There has been little research done on systems that could solve the ‘odd man out’ problem, however, in a literature paper about computer modules solving odd one out questions [1], there was a brief section on potentially using categories to differentiate between objects. This concept could be a possible path to solving text based odd man out questions.

There have been many other attempts on solving a variety of odd one out questions, such as utilising a robot to find characteristics of an object, to determine the odd one out [3]. There has also been research on testing how well online search engines perform on custom IQ test questions [4].

Recently, there has been major breakthroughs with machine learning and deep neural networks. A prime example of this is AlphaGO [6]. By utilizing and adapting these new methods, and applying them to solve IQ test questions, there could possibly be a major breakthrough in perfecting solving the ’selecting the odd one out problem’

2.2 Research

The only existing research that helped enable the project to happen were two papers. One about a current computer model that is able to solve certain IQ test questions [1], which gave me ideas about using categories to differentiate between given input items.

And another paper on how utilising the internet to assist in solving questions [4]. This paper revolved around using a search engine to answer questions. The concept of using the Internet was pulled over to this project, where I employ an online database to obtain necessary data to solve odd one out questions, instead of creating my own database.

2.3 Dataset used

For the project to work, it would require a massive collection of data, on as many words as possible. For each word found within the dataset, there must be a set of words that categorize the data. An example of what the data may look like can be seen below for the word ‘Mars’ in Figure 1.

The data used for this project is obtained from Wikipedia using an open source software package called MediaWiki.

`https://www.mediawiki.org/wiki/API`

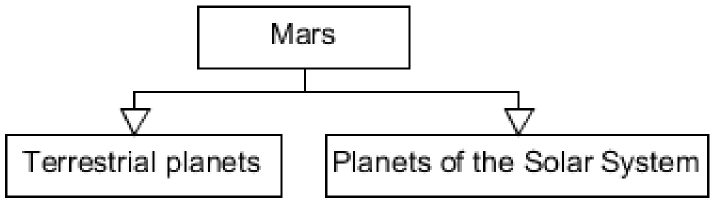


Figure 1: Example of data retrieved from Wikipedia.

MediaWiki has a query API (application programming interface) that allows users to obtain data directly from pages on Wikipedia, therefore allowing us to obtain the necessary data for each word inputted into the program. The program uses this API because it allows us to obtain the required data fast and efficiently. It is efficient because it allows us to string together multiple queries, which increases our program speed and reduces loads on Wikipedia's servers.

2.4 Project Risks

There are two main project risks involved, both risks were analyzed during project specification [5] and are shown below in section 2.4.1 and 2.4.2. Both these risks can heavily cripple the program, but have very low probabilities of happening and can be planned for.

2.4.1 Internet Access Down

The program requires Internet to access Wikipedia's API to obtain the required data. If there is no Internet access, or Wikipedia is down, the program will not work.

The probability of this risk occurring is generally low, as Wikipedia has a solid uptime. However, if you are in an area prone to Internet shortages then the probability increases. For an average person the probability would be approximately 5%.

The risk can be reduced by ensuring you are connected to a solid Internet during runtime. This risk can be solved by having an offline database stored locally, and have the program fetch data from that. However, in doing this, the program will miss out on any updates to the dataset. And will also require lots of storage for the program.

2.4.2 MediaWiki modifies API

When the program utilizes MediaWiki's query API to access data from Wikipedia, it follows certain methods and parameters set out by Wikipedia. If for some reason, any of these are changed without prior warning, the program will cease to work indefinitely unless it is patched.

The probability of this risk is extremely low, as there are nearly no reasons to change the API methods.

Since most, if not all companies, will make the call 'deprecated' we will have time to create a new version of our program before the API call becomes completely redundant. Thereby allowing the risk to be mitigated.

2.4.3 Users input item not found in dataset

Since all user inputs are checked on Wikipedia for data, if a user input item does not have a page on Wikipedia then the program can not proceed. Although Wikipedia hosts a massive amount of data, there possibly may be some missing pages on a user's input.

The probability of this risk happening is low, around 10%. During all my testing the only times when I received an error about a page not existing is when I purposefully entered a typo.

This risk can be solved by having the program inform the user that the word could not be found, then gracefully exiting. Making sure the user understands why the program exited.

2.4.4 Users input ambiguous item

When a user enters an item, it is checked for data on Wikipedia. However sometimes the user inputs an item that is ambiguous, meaning that it could be referring to multiple items. Example the item 'table' could be referring to 'table spreadsheet' or 'table furniture'. In this case Wikipedia will display a page showing alternative words the user could be meaning. If our program cannot handle this input, then the output will either be incorrect or the program may crash.

The probability of this risk is high, around 60%. As during testing there were many ambiguous words found.

This risk can be solved by retrieving the alternative inputs Wikipedia displays, then giving the user an option to select the correct meaning.

2.4.5 Risks Encountered

During the implementation process of the project, the only risk I ran into unintentionally was the 'user inputting ambiguous item', which crashed the program. This allowed me to develop code stop this from happening again by asking for clarification from the user. There were no external issues that caused any harm to my schedule and project. The risk of 'Internet Access Down' only occurred when I purposely disconnected my device from the Internet to test if my program exits gracefully during an outage. And I also purposefully inputted a non-existing word to simulate the risk of 'Users input item not found in dataset' risk to ensure my program also fails gracefully from that.

2.5 Resource Requirements

The resource requirements changed between the project specification [5] and implementation. A new requirement 'Python module Reqeusts' was added. The other resource requirements were taken from the project specification [5].

2.5.1 Python v3.6.4

The whole program is written in and run with Python v3.6.4. The development of the program will be done within 'PyCharm: Python IDE by JetBrains'. Python was chosen as it had all the necessities needed for this program, such as API calls to the Internet. The program will definitely not run on any versions below 3.0. And may potentially run on versions between 3.6 and 3.0. However these versions have not been tested.

2.5.2 Python module 'Requests'

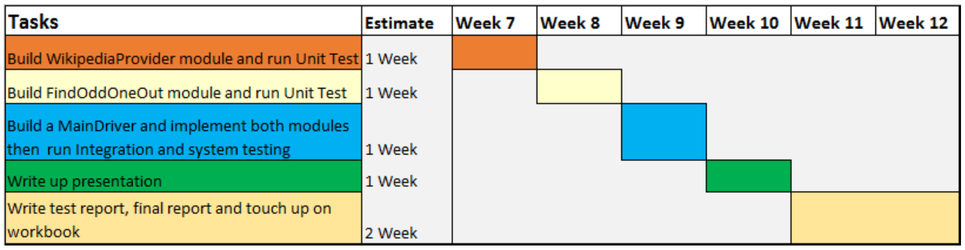
Since the program required to retrieve data from online sources, I needed a module for Python called 'requests' to handle that. Requests was selected because it was fast, and simple to use.

2.5.3 Computer

The only requirement for the computer is that it has Internet access, and Python v3.6 or above installed. Versions between v3.0 and v3.6 may potentially work and any version below 3 will definitely fail. The program is not hardware intensive, so it will be able to run on low end computers.

2.6 Project Schedule

The general schedule of the project is shown in Figure 2 below, sourced from the project specification [5]. The project successfully ran on time with no delays. There were no issues during the process, and if any did occur I ensured there was a buffer to allow the project to get back on track. The buffer is the ‘Write up presentation’ task which can easily be completed in less time, thereby giving me more time to do other tasks if need be.



Tasks	Estimate	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Build WikipediaProvider module and run Unit Test	1 Week	🟧					
Build FindOddOneOut module and run Unit Test	1 Week		🟨				
Build a MainDriver and implement both modules then run Integration and system testing	1 Week			🟦			
Write up presentation	1 Week				🟩		
Write test report, final report and touch up on workbook	2 Week					🟨	🟨

Figure 2: Gantt Chart of project.

3 Method

3.1 Methodology

The methodology of this project is to use categories to find relations between a given set of items. If we take these items, then create all possible combination sets of items, sized (number of items - 1). We can use categories of each item to determine which other items share the same categories. If two items or more items share the same category, it indicates that they have a relation. And if there are common categories between (number of items - 1) items. That would mean that the other item not included is the odd one out.

To actually bring this method to life, it will require a very large set of data that contains information about items categories. Therefore we will be using Wikipedia’s massive collection of categories, which will be accessed through MediaWiki’s query API.

3.2 Internal design

The completed project has two main modules to obtain, filter and analyse data. There is also a MainDriver module that ties together these modules. A sequence diagram can be seen below describing the process of a user using the program in Figure 3.
The MainDriver will obtain the input from a user, validate it, then send the items to WikipediaProvider. The WikipediaProvider is tasked with requesting data from MediaWiki's query API to obtain information on each input given to it. After an initial search, data is send back to the MainModule.
The MainDriver module will validate the inputs again, and request more data if necessary. The obtained data is now sent to the FindOddOneOut module to determine the odd one out. It will send the results back to MainDriver after it is completed, where the results will be displayed.

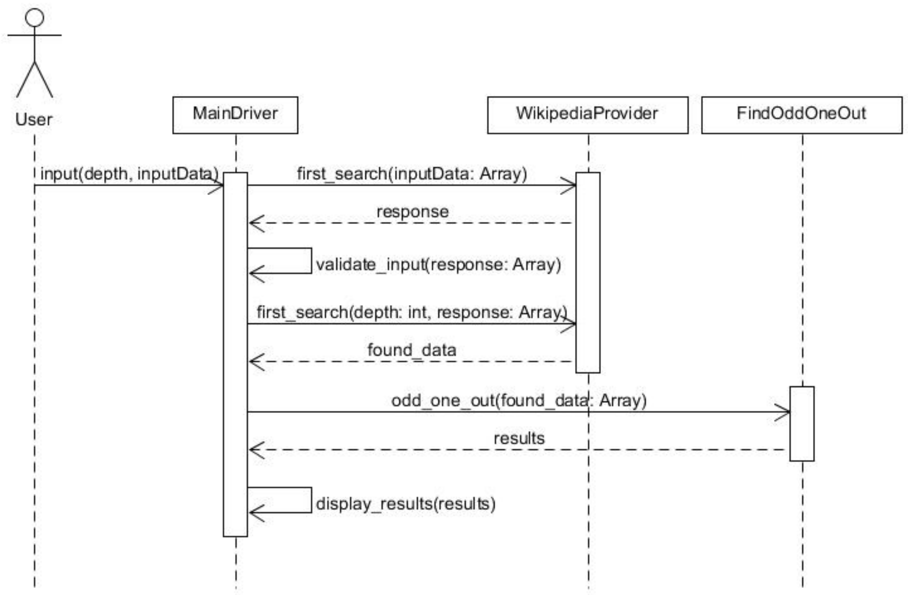

User
MainDriver
WikipediaProvider
FindOddOneOut
input(depth, inputData)
first_search(inputData: Array)
response
validate_input(response: Array)
first_search(depth: int, response: Array)
found_data
odd_one_out(found_data: Array)
results
display_results(results)

Figure 3: Sequence diagram of project.

3.3 Software Architecture

The overall structure of this computer program is relatively simple. There are 3 classes, which can be seen below in Figure 4, with the general structure, methods and attributes of the program. The general roles of each module is:

- WikipediaProvider - Obtain data from online database
- FindOddOneOut - Determine odd one out from inputted data
- MainDriver - Tie modules together, receive and validate input, display results

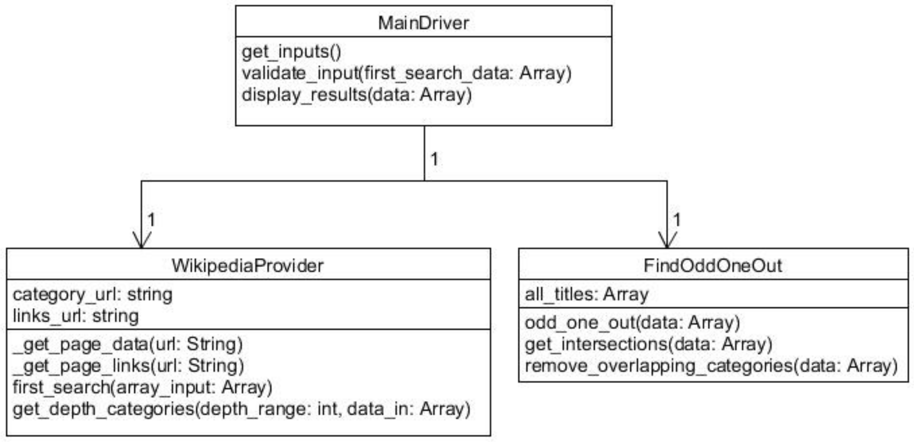
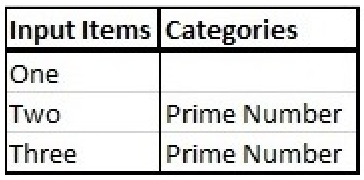

MainDriver
get_inputs()
validate_input(first_search_data: Array)
display_results(data: Array)
1
1
1
WikipediaProvider
category_url: string
links_url: string
_get_page_data(url: String)
_get_page_links(url: String)
first_search(array_input: Array)
get_depth_categories(depth_range: int, data_in: Array)
FindOddOneOut
all_titles: Array
odd_one_out(data: Array)
get_intersections(data: Array)
remove_overlapping_categories(data: Array)

Figure 4: Class diagram of project.

3.4 Key algorithms

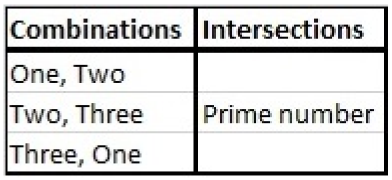
The premise of this program is very basic. The most difficult part of it is obtaining the data recursively, then also cleaning it up into usable data structures.

A key function within the 'FindOddOneOut' Module is the 'get_intersections()' function. This function obtains all possible (n - 1) sets of inputted words, and gets the intersections of the categories for each set of words. For example, given list of inputs and categories seen in Figure 5, you can generate a list of combinations of inputs sized (number of inputs - 1). This would generate the combinations seen in Figure 6's first column. And by getting the intersections of categories, we can determine the the combination 'Two' and 'Three' have a common category 'Prime number'. Therefore 'One' is the odd one out.



Input Items	Categories
One	
Two	Prime Number
Three	Prime Number

Figure 5: Sample Data



Combinations	Intersections
One, Two	
Two, Three	Prime number
Three, One	

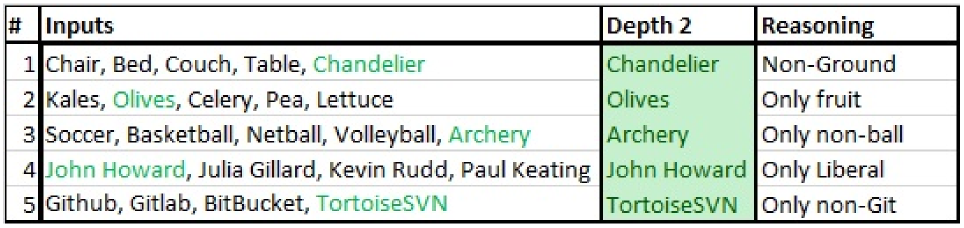
Figure 6: Categories intersect

4 Results

The results for this computer program have exceeded what I thought could obtain. I ran this computer program on many questions and have allowed others to do the same.

4.1 Results from test questions

These questions were developed early on in the implementation stage as test questions. The program had successfully gotten all of them right as seen below in Figure 7.



#	Inputs	Depth 2	Reasoning
1	Chair, Bed, Couch, Table, Chandelier	Chandelier	Non-Ground
2	Kales, Olives, Celery, Pea, Lettuce	Olives	Only fruit
3	Soccer, Basketball, Netball, Volleyball, Archery	Archery	Only non-ball
4	John Howard, Julia Gillard, Kevin Rudd, Paul Keating	John Howard	Only Liberal
5	Github, Gitlab, BitBucket, TortoiseSVN	TortoiseSVN	Only non-Git

Figure 7: Output data formatted

4.2 Results from an on-line Quiz

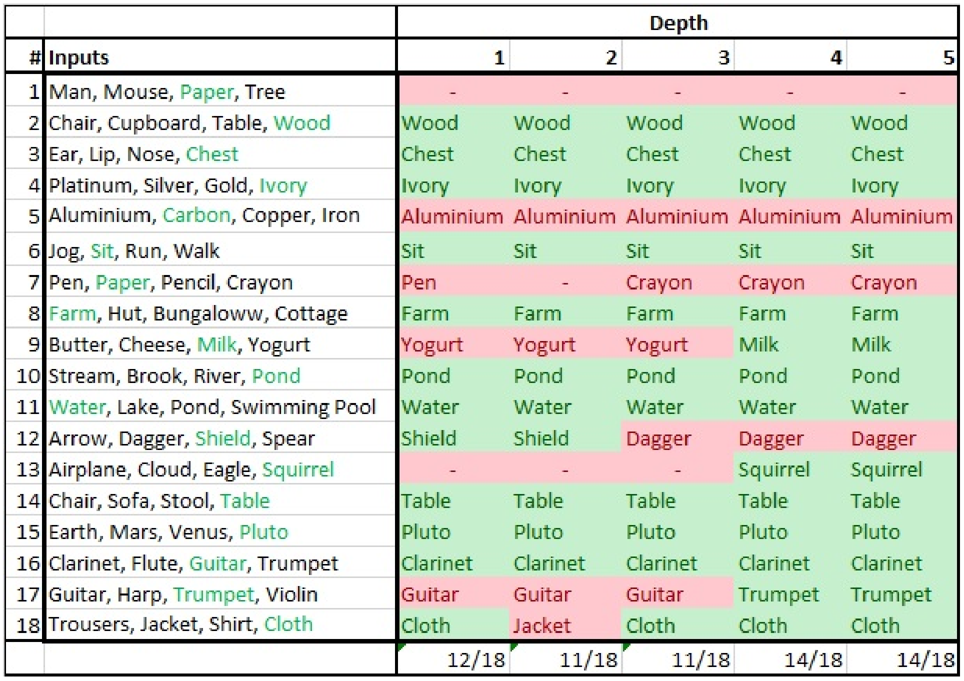
Test data for IQ select the odd one out questions were sourced from a website to be inputted into the program. The questions were sourced from

`https://www.syvum.com/cgi/online/serve.cgi/iq/ar_oddo2.html?table`

I had to slightly modify some questions due to the quiz being old, and having questions I already determined will not be handled, seen in section 1.3 labeled

‘Constraints’. Therefore I removed 3 questions that related to dates and pronunciation. I also modified question number 15 shown below in Figure 8 by replacing ‘Moon’ with ‘Venus’. Because prior to this change, the question assumed that ‘Pluto’ is a planet, which it is not anymore.

At the depth search of 5, the program was already taking up approximately 10 seconds (depth * 2) per question. Therefore I decided to only test up to a search depth of 5.



		Depth				
#	Inputs	1	2	3	4	5
1	Man, Mouse, Paper, Tree	-	-	-	-	-
2	Chair, Cupboard, Table, Wood	Wood	Wood	Wood	Wood	Wood
3	Ear, Lip, Nose, Chest	Chest	Chest	Chest	Chest	Chest
4	Platinum, Silver, Gold, Ivory	Ivory	Ivory	Ivory	Ivory	Ivory
5	Aluminium, Carbon, Copper, Iron	Aluminium	Aluminium	Aluminium	Aluminium	Aluminium
6	Jog, Sit, Run, Walk	Sit	Sit	Sit	Sit	Sit
7	Pen, Paper, Pencil, Crayon	Pen	-	Crayon	Crayon	Crayon
8	Farm, Hut, Bungaloww, Cottage	Farm	Farm	Farm	Farm	Farm
9	Butter, Cheese, Milk, Yogurt	Yogurt	Yogurt	Yogurt	Milk	Milk
10	Stream, Brook, River, Pond	Pond	Pond	Pond	Pond	Pond
11	Water, Lake, Pond, Swimming Pool	Water	Water	Water	Water	Water
12	Arrow, Dagger, Shield, Spear	Shield	Shield	Dagger	Dagger	Dagger
13	Airplane, Cloud, Eagle, Squirrel	-	-	-	Squirrel	Squirrel
14	Chair, Sofa, Stool, Table	Table	Table	Table	Table	Table
15	Earth, Mars, Venus, Pluto	Pluto	Pluto	Pluto	Pluto	Pluto
16	Clarinet, Flute, Guitar, Trumpet	Clarinet	Clarinet	Clarinet	Clarinet	Clarinet
17	Guitar, Harp, Trumpet, Violin	Guitar	Guitar	Guitar	Trumpet	Trumpet
18	Trousers, Jacket, Shirt, Cloth	Cloth	Jacket	Cloth	Cloth	Cloth
		12/18	11/18	11/18	14/18	14/18

Figure 8: Output data formatted

Cells that are completely green are correct, whereas cells that are colored red are incorrect. The correct answer for any question is seen under the inputs column and the word highlighted green.

4.3 Externally observable features

4.3.1 Input

The program will immediately take 3 main inputs from the user when the program is run. Also during runtime, the program may also ask for more inputs. Details on the search depth parameter are sourced from the project specification [5].

Number of inputs This defines how many items can be inputted. The minimum is 3, because logically speaking you cannot select the odd one out with two items or less. The maximum number of inputs is capped at 19. Because we do not want to overburden MediaWiki's query API with too many requests.

Question input This is the actual items the user inputs for the program to determine the odd one out. For example, a user could input:

1. Chair
2. Table
3. Cupboard
4. Apple Pie

If previously they have set the 'number of inputs' parameter to 4.

Search depth This is how deep the program will search through the categories as shown below in Figure 9. Increasing the depth increases the program running time by approximately (depth*2) seconds. The deeper you go, the more categories describing your search item is found. A search depth of 2 is the recommended setting, because after 2, the categories start to become unfocused. The minimum value is 1. And the maximum value is 10, to ensure we do not overburden Wikipedia's servers.

Each time you go down a depth you get categories that describe the previous categories. As seen in Figure 9.

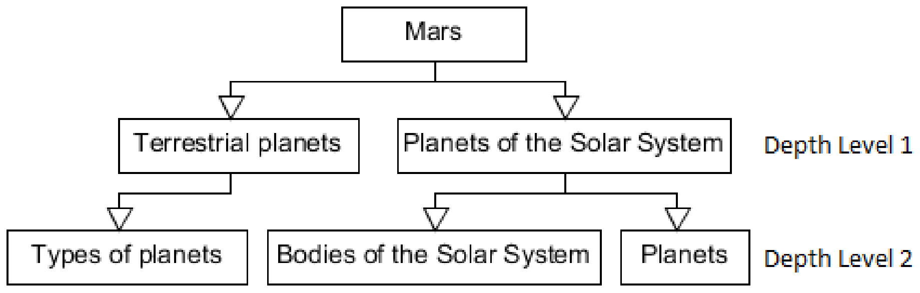
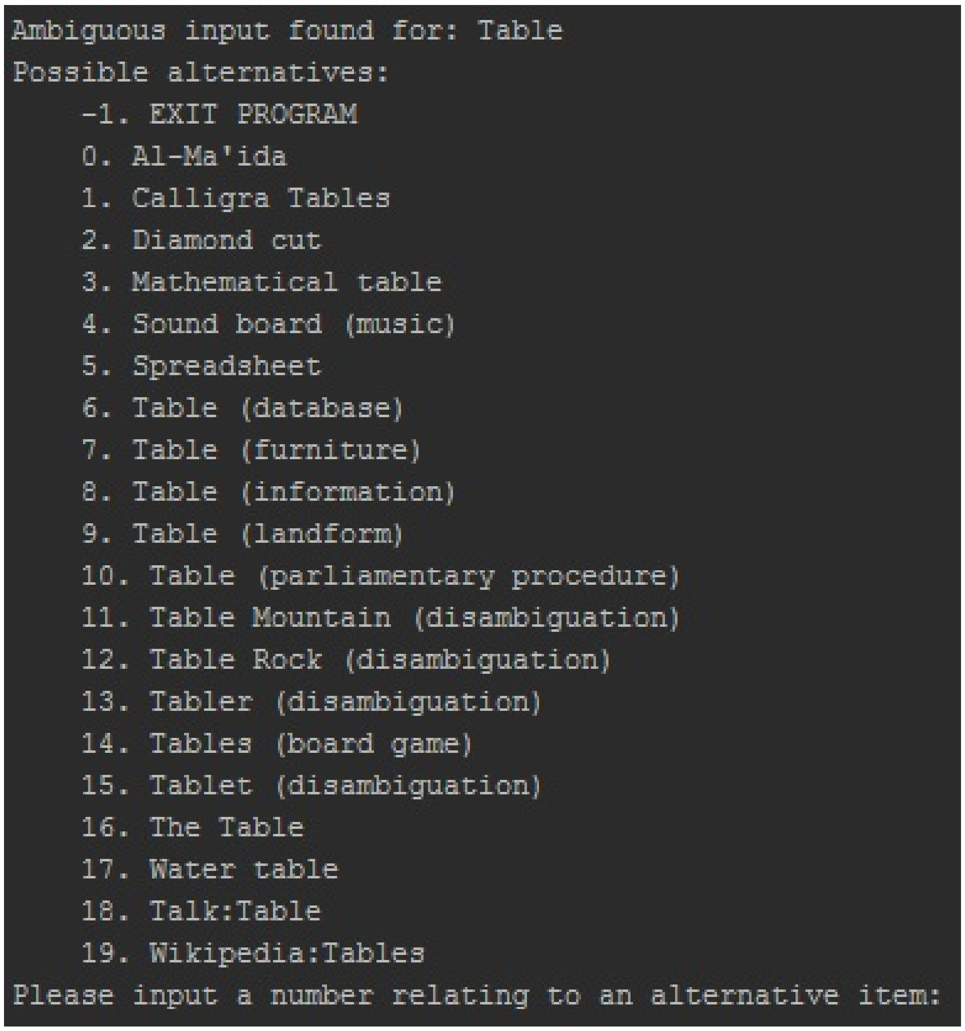


Figure 9: Example of categories found when searching for 'Mars'.

Clarification input During program runtime there may be a moment the program may ask for clarification on an input. The user will be shown a list of items to choose from and are requested to select one by the index shown next to the item to proceed. This can be seen occurring below in Figure 10.



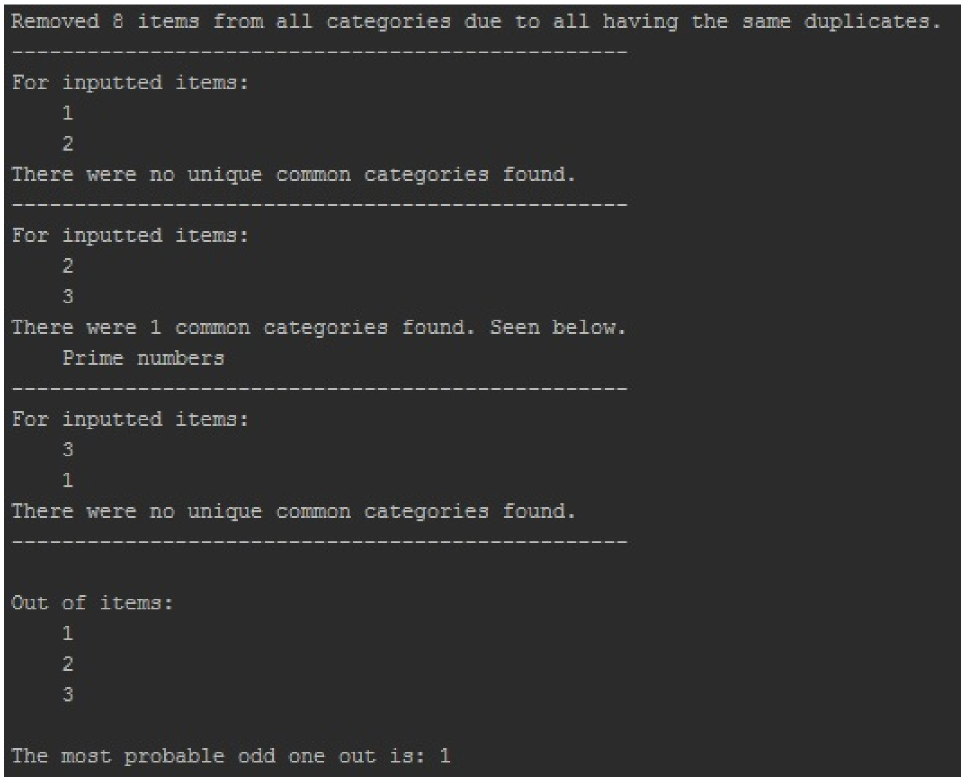
```
Ambiguous input found for: Table
Possible alternatives:
    -1. EXIT PROGRAM
    0. Al-Ma'ida
    1. Calligra Tables
    2. Diamond cut
    3. Mathematical table
    4. Sound board (music)
    5. Spreadsheet
    6. Table (database)
    7. Table (furniture)
    8. Table (information)
    9. Table (landform)
    10. Table (parliamentary procedure)
    11. Table Mountain (disambiguation)
    12. Table Rock (disambiguation)
    13. Tabler (disambiguation)
    14. Tables (board game)
    15. Tablet (disambiguation)
    16. The Table
    17. Water table
    18. Talk:Table
    19. Wikipedia:Tables
Please input a number relating to an alternative item:
```

Figure 10: Ambiguous Input

4.3.2 Output

There is no graphical user interface, as the program is terminal based. After the program has attempted to try determine the output, it will print out the results. An example can be seen below in Figure 11.

Note that the first printed line 'Removed 8 items...' indicates that all 3 items (1, 2, 3) had 8 common categories. Therefore it will not be necessary to have them shown because they are not unique, and will have no affect on the outcome. The program also shows that items 2 and 3 have a common category 'Prime numbers' which item 1 does not have. Therefore it must be the odd one out, which is displayed at the end of the picture.



```
Removed 8 items from all categories due to all having the same duplicates.
-------------------------------------------------
For inputted items:
    1
    2
There were no unique common categories found.
-------------------------------------------------
For inputted items:
    2
    3
There were 1 common categories found. Seen below.
    Prime numbers
-------------------------------------------------
For inputted items:
    3
    1
There were no unique common categories found.
-------------------------------------------------

Out of items:
    1
    2
    3

The most probable odd one out is: 1
```

Figure 11: Program Output

4.4 Performance

Real-world Processing time Since the computer program requests data from online servers, there will be a delay in sending the request, and receiving that data. On a normal Internet connection one request may take on average 2 seconds. Since the program can recursively obtain more data due to the input parameter ‘search depth’, the running time of the computer program can vary. Therefore on average the computer program will take (2 * n) seconds to produce an answer, where n is the ‘search depth‘. Also if there is an ‘ambiguous’ input where the program asks the user for clarification, the program will have to do another 2 API calls, adding 4 seconds onto the program running time. Because the API calls takes so much time, the code’s complexity does not affect the running time of the program at all. Regardless, the big O complexity of the code is $O(n^2)$, where n is the number of inputs.

Storage space The storage this program takes up is insignificant due to just being text based. It weighs in to less than 50 kilobytes. However for the program to run it requires the package ‘requests’ which will be downloaded. And is approximately 350 kilobytes. Overall the computer program will definitely stay below the 1 megabyte mark.

Bandwidth Since the computer program is only pulling a extremely minuscule amount of data from Wikipedia’s servers, the bandwidth it requires will not be large. As long as the computer running the program has a stable Internet access, any bandwidth will do.

5 Analysis and Discussion

5.1 Analysis of test question results

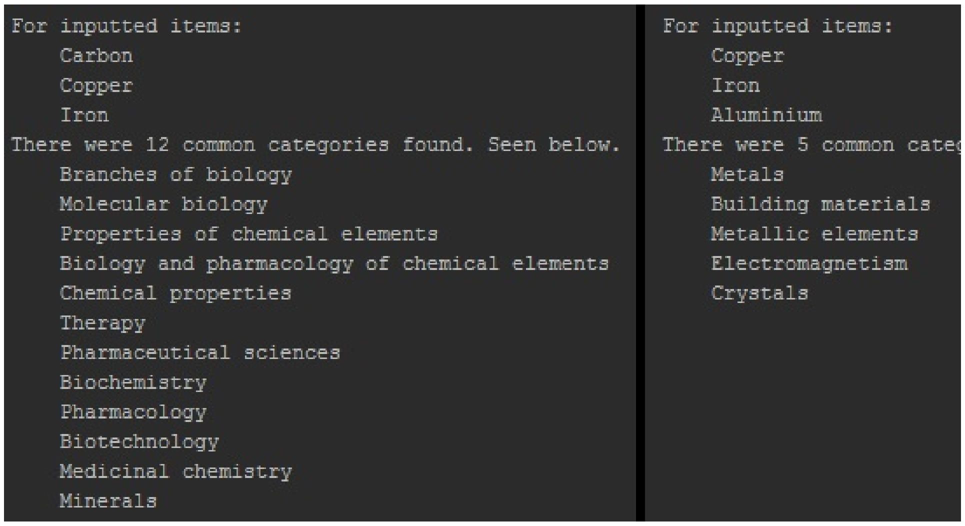
The initial test questions seen in Figure 7 all ran successfully without any problems. The results from this show the large variety of questions the computer program can handle.

5.2 Analysis of online quiz results

The results of from the online quiz seen in Figure 8 shows that the program doesn’t always work 100% of the time. There are three questions that the program got wrong for all depths. Questions 1, 5 and 7 are consistently wrong.

All dashes in results For question 1 there were simply no common unique categories found between any 3 combination of inputs, therefore the program couldn't determine the odd one out.

Question 5 For question 5, the correct answer is 'Carbon' because it is the only gas. However the program kept outputting 'Aluminium'. This is because there are simply more common categories between 'Carbon, Copper and Iron' than 'Copper, Iron and Aluminium'. The common categories can be seen below in Figure 12.



```
For inputted items:
    Carbon
    Copper
    Iron
There were 12 common categories found. Seen below.
    Branches of biology
    Molecular biology
    Properties of chemical elements
    Biology and pharmacology of chemical elements
    Chemical properties
    Therapy
    Pharmaceutical sciences
    Biochemistry
    Pharmacology
    Biotechnology
    Medicinal chemistry
    Minerals
```

```
For inputted items:
    Copper
    Iron
    Aluminium
There were 5 common cate
    Metals
    Building materials
    Metallic elements
    Electromagnetism
    Crystals
```

Figure 12: Program Test Output

This illustrates that even if there is an obvious correct common category, 'Metals' that link the correct items together, the program will choose the set of items with the highest number of unique common categories. This is a flaw that could potentially be fixed by giving more common words like (metal), a higher weight.

Answer changing between search depths Question 12 in Figure 8 changes answer midway due to finding more categories relating to the other item combinations. This happened because in my opinion, that question has multiple answers. You could say 'Arrow' is the only ranged weapon, therefore is the odd one out. Or you could say that 'Dagger' is the most currently used modern weapon out of the set of items. 'Shield' is also the only defensive weapon. Therefore I would say this is a poor question and shouldn't be used as an actual test question.

No answers for first few depths There are also times where the first few depths do not obtain an answer at all. This is due to the items in the question being so far apart in the dataset, that it takes recursively searching a high number of times to find common categories.

A dash output In the results sometimes there is a dash, when clearly in a previous search depth there was an answer found. This is because sometimes there are 2 or more sets of items that have the exact same number of categories. Therefore the program is unable to differentiate between the correct and wrong answers.

5.3 Discussion

The first test resulted in a 100% success rate. Whereas the second test, which used online test questions, achieved a highest score of 78% with a search depth of 5. There are no results of other machines that I am able to compare with, as these types of odd one out questions are rarely visited with computer models. People have thought of methods to complete them [1] but have never actually implemented them. This may be because the ‘odd one out’ type questions are potentially being phased out of IQ test questions.

6 Future Work

With concept of using categories to solve odd-one-out questions being completed, it has paved a way for possible future work to be done on this topic.

6.1 Store data locally

The project’s real time speed is capped by the time it takes to obtain data from Wikipedia’s online database. If theoretically, we stored Wikipedia’s whole database locally, it could cut down the programs run time from seconds to milliseconds. Since Wikipedia allows individuals to actually store their data to preserve free information, they have set up data dumps for people to download. It is possible to create a new module to instead of running API calls, to run calls locally to obtain that data.

6.2 Update categories on Wikipedia

Theoretically since all users can edit Wikipedia, we could update categories fix common questions, to allow for better results in our computer program. However this is time consuming and inefficient.

6.3 Create Odd-One-Out Questions

If we flip the current program's concept, and instead of using categories to solve questions, we could generate odd one out questions ourselves. An example question is shown below in Figure 13. An example of how to generate this question is to first, select a random item, example Harps. Then go one level up to its categories, 'String instrument' and selecting two other items within that category, 'Violins' and 'Zither'. Then by going up one level, and choosing another unrelated category 'Jazz instruments' we can then choose one item from that category 'Saxophone'.

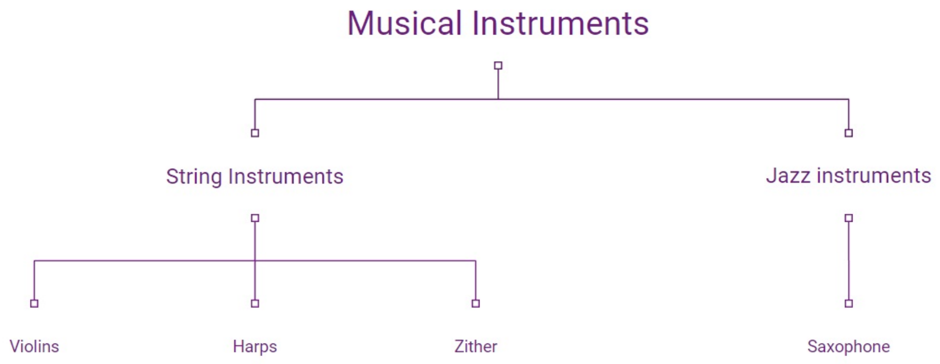


Figure 13: Generating an IQ odd one out question

7 Conclusion

Computer programs attempting to solve IQ test questions have been happening for decades. But with this burst of renewed interests to solve these types of questions, this computer program was developed specifically to join in the race to solve IQ test questions. Using the concept of categories, this project achieved a relatively high score for selecting the odd one out questions. However I expect in the future, there will be people to combine the techniques shown here, and machine learning techniques, to create an improved version of this concept.

References

[1] P. Sanghi and D. L. Dowe, “A computer program capable of passing iq tests,” in *4th Intl. Conf. on Cognitive Science (ICCS’03), Sydney*, pp. 570–575, 2003.

[2] B. Fernando, H. Bilen, E. Gavves, and S. Gould, “Self-supervised video representation learning with odd-one-out networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5729–5738, IEEE, 2017.

[3] J. Sinapov and A. Stoytchev, “The odd one out task: Toward an intelligence test for robots,” Aug 2010.

[4] F. Liu and Y. Shi, “The search engine iq test based on the internet iq evaluation algorithm,” *Procedia Computer Science*, vol. 31, pp. 1066–1073, 2014.

[5] D. Nguyen, “Solving odd-one-out iq test problems with a computer model,” 2018.

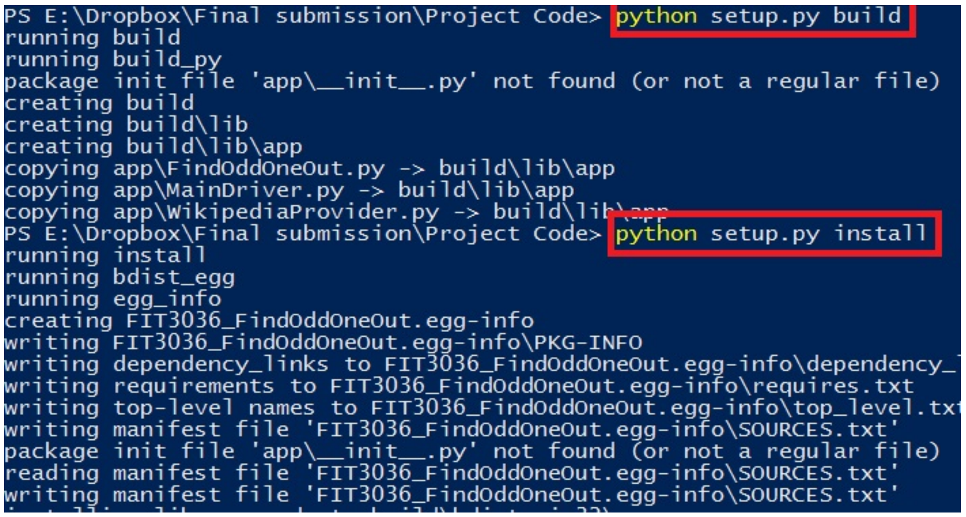
[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

A
Program Installation

The program requires dependencies to run. Therefore, to install the dependencies for this program you are required to open a terminal on a machine (Windows and Linux have been tested) with Python 3.6 and above. You may need to run the terminal with administrator privileges to install the module.

1. Open the terminal on your PC
2. Head into the ‘Project Code’ folder inside your terminal and input the commands below into the terminal
3. python setup.py build
4. python setup.py install

Some reference pictures are shown at Figure 14 below for running commands 3 and 4 listed above. Note if you are having problems and/or you are seeing python version 2 show up, but know that you have version 3 and above installed, try to use the command python3 instead of python.



```
PS E:\Dropbox\Final submission\Project Code> python setup.py build
running build
running build_py
package init file 'app\__init__.py' not found (or not a regular file)
creating build
creating build\lib
creating build\lib\app
copying app\FindOddOneOut.py -> build\lib\app
copying app\MainDriver.py -> build\lib\app
copying app\WikipediaProvider.py -> build\lib\app
PS E:\Dropbox\Final submission\Project Code> python setup.py install
running install
running bdist_egg
running egg_info
creating FIT3036_FindOddOneOut.egg-info
writing FIT3036_FindOddOneOut.egg-info\PKG-INFO
writing dependency_links to FIT3036_FindOddOneOut.egg-info\dependency_
writing requirements to FIT3036_FindOddOneOut.egg-info\requires.txt
writing top-level names to FIT3036_FindOddOneOut.egg-info\top_level.tx
writing manifest file 'FIT3036_FindOddOneOut.egg-info\SOURCES.txt'
package init file 'app\__init__.py' not found (or not a regular file)
reading manifest file 'FIT3036_FindOddOneOut.egg-info\SOURCES.txt'
writing manifest file 'FIT3036_FindOddOneOut.egg-info\SOURCES.txt'
```

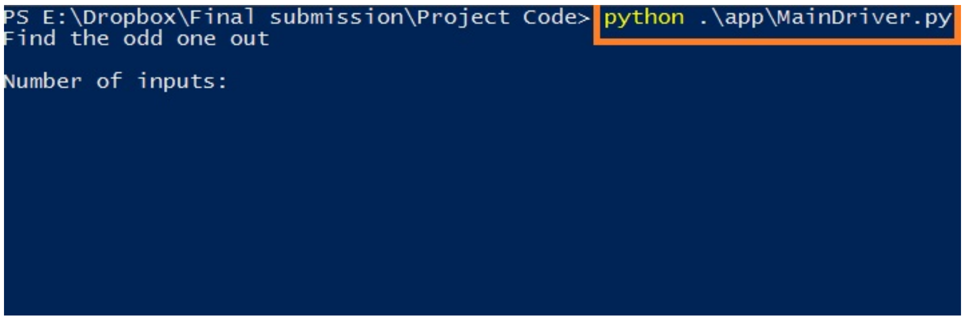
Figure 14: Inputting commands to get dependencies

B
Running Program

To run this program, you will have been required to install the dependencies shown in Appendix A. After doing so, continue these steps to start the program.

1. Head to the program directory where you can see the setup.py file. Then input the line below into the terminal (make sure you are in the Program Code folder within the terminal).
2. python app/MainDriver.py

Now you should be greeted with an interface shown below in Figure 15.



```
PS E:\Dropbox\Final submission\Project Code> python .\app\MainDriver.py
Find the odd one out

Number of inputs:
```

Figure 15: Running the program

Example runthrough of program Lets assume we want to determine the odd one out of these items listed below. (Pie is odd one out)

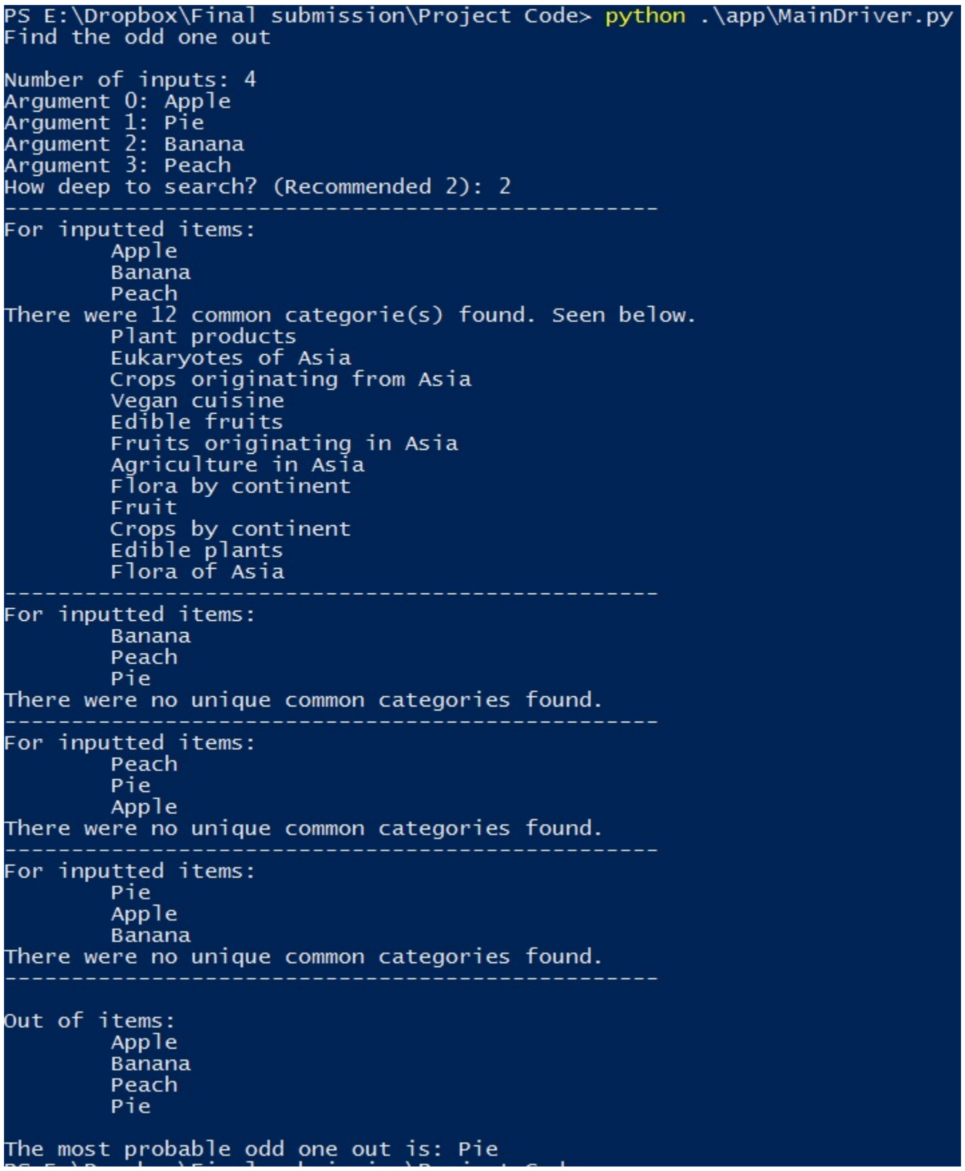
- Apple
- Banana
- Peach
- Pie

Steps to input it into the program:

1. Launch the program as shown above
2. For the Number of inputs field, enter 4
3. Now enter, one line at a time, the 4 items above (Apple, Banana, Peach, Pie)
4. You should now see a “How deep to search?” input field. Input the number 2 in and press enter.

5. The program will now obtain the necessary data from online, and display you the output in a few seconds.

This whole process can be seen down below in Figure 16.



```
PS E:\Dropbox\Final submission\Project Code> python .\app\MainDriver.py
Find the odd one out

Number of inputs: 4
Argument 0: Apple
Argument 1: Pie
Argument 2: Banana
Argument 3: Peach
How deep to search? (Recommended 2): 2
-------------------------------------------------
For inputted items:
        Apple
        Banana
        Peach
There were 12 common categorie(s) found. Seen below.
        Plant products
        Eukaryotes of Asia
        Crops originating from Asia
        Vegan cuisine
        Edible fruits
        Fruits originating in Asia
        Agriculture in Asia
        Flora by continent
        Fruit
        Crops by continent
        Edible plants
        Flora of Asia
-------------------------------------------------
For inputted items:
        Banana
        Peach
        Pie
There were no unique common categories found.
-------------------------------------------------
For inputted items:
        Peach
        Pie
        Apple
There were no unique common categories found.
-------------------------------------------------
For inputted items:
        Pie
        Apple
        Banana
There were no unique common categories found.
-------------------------------------------------

Out of items:
        Apple
        Banana
        Peach
        Pie

The most probable odd one out is: Pie
```

Figure 16: Running the program with an example input