# Accredited Vacation Project

## Section 1: Summary of Work Placement

Over the summer I spent three months working at a company called MeVitae. MeVitae's aim is to make the hiring process more efficient for recruiters. Specifically, when shortlisting candidates from CV's. MeVitae plans to conquer this aim by creating an API that a collection of CVs and a job description can be sent to. The API will then return the CVs, each with a score based on its content in comparison to the job description content. A recruiter can then shortlist the candidates based on the CV scores.

MeVitae is a start-up and the whole company is a single department. The team comprised of a CEO, a CTO and an Algorithm Development Manager. Also, another intern and me. Everybody worked in the same room, this created a great dynamic for bouncing ideas of each other or asking for help. Whiteboards were all over the walls which I greatly appreciated. It enabled me and others to explain and teach the things we had been working on to the rest of the team, as sort of mini-lectures.

I was employed as a software developer intern. I was there to help write the code and algorithms for the API. The algorithms and structure of the API had already been planned out by the CTO and Algorithm Developer They explained the plan to me and the other intern on the first day and throughout the internship. The plan was not set in concrete though and any ideas I had were welcomed if I could prove they were promising.

The internship was set into three stages: the research phase, the pre-development phase and development phase. The research phase was an opportunity for me to learn the techniques we planned to use and to teach myself a foundation level of knowledge on machine learning and natural language processing. Each week I was given a list of tasks. These tasks involved implementing and testing algorithms and reading research papers. The pre-development phase involved me using the data MeVitae had gathered and completing tasks (which I will go into more detail later) with it to try and get promising results. The algorithms and code that gave promising results would then be tested further and possibly used in the API. The development phase was where we started to build the API and putting together what everyone had been working on together. Originally, I would not have been a part of the development phase as the internship was planned to be 2 months long. MeVitae extended my stay by one month because I proved my ability to be an effective employee.

## Section 2: Aims and Objectives

I was told my aims on the first day along with the explanation of the plan of development. The first was to understand the basics of machine learning and natural language processing. This was set so that I would understand how the API would work and how what I was doing would fit in with the bigger picture. Another aim was for me to create a tree data structure to store and represent all the Wikipedia data they had scrapped. In the tree, the data could then be visualised and manipulated to create training data for a recurrent neural network.

My first aim was to increase my knowledge of machine learning and see if it was something I enjoyed and was interested in. My second aim was to create production worthy code and contribute to the start-up in a high impact way. My third aim was to come up with an original idea that would influence the MeVitae's final product. Lastly, I wanted to learn how things run in a start-up and what the development process is like and decide if it is a work environment for me in the future.

## Section 3: Tasks Completed and experience gained

The work cycle was structured in the following way:

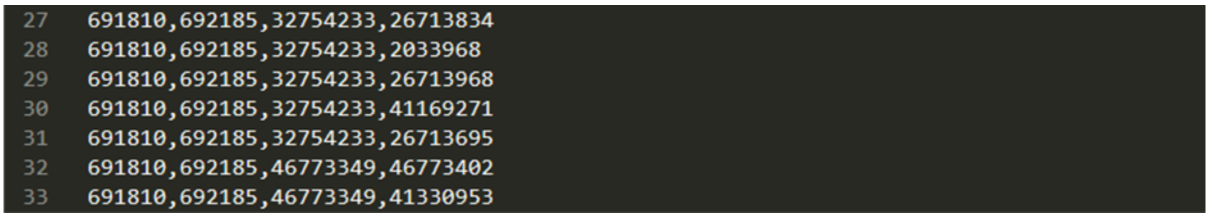
- At the start of the week (Monday) a task would be given, and we would discuss different options for how I could complete it.
- On Friday, I would explain to everyone what I had done and how I did it through a short lecture, including me drawing diagrams on the whiteboard. I found doing this really helped cement my knowledge of what I was doing and forced me to learn things in detail as I knew I would have to explain it.

The first 3 weeks of the internship involved me researching various topics. I could explain extensively into each topic I researched and tested but I wish to save the deep explanations for the actual contributions I gave to the company. Instead, I will simply list the researched topics in chronological order grouped by week:

- N-Grams, Word2Vec [4], PCA
- Convolutional and Recurrent Neural Network in CNTK,
- K-Means Clustering, Markov Chains, Ada Boost

The main task I was given after researching the techniques used in the current stack, was to build a tree-like data structure. The tree held all the Wikipedia data that had been scrapped. The data was 1.4 Million categories from the academic disciplines section of Wikipedia. The category paths were stored in text files. Each line in the text file contained the list of ID's representing the path to category ID at the end. In figure 1 line 27, '26713834' is the category and the list of ID's before it is the path to it on Wikipedia. This also means that '691810' is the top category and each ID after in it the lines are a jump down into a subcategory.

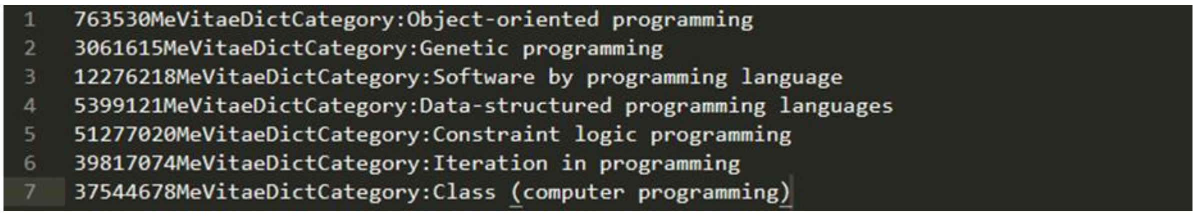
**Figure 1**



```
27  691810,692185,32754233,26713834
28  691810,692185,32754233,2033968
29  691810,692185,32754233,26713968
30  691810,692185,32754233,41169271
31  691810,692185,32754233,26713695
32  691810,692185,46773349,46773402
33  691810,692185,46773349,41330953
```

A separate text file contained each category ID and their name. I read in this file and used it as a lookup table for finding a categories name using the ID. (See Figure 2)

**Figure 2**

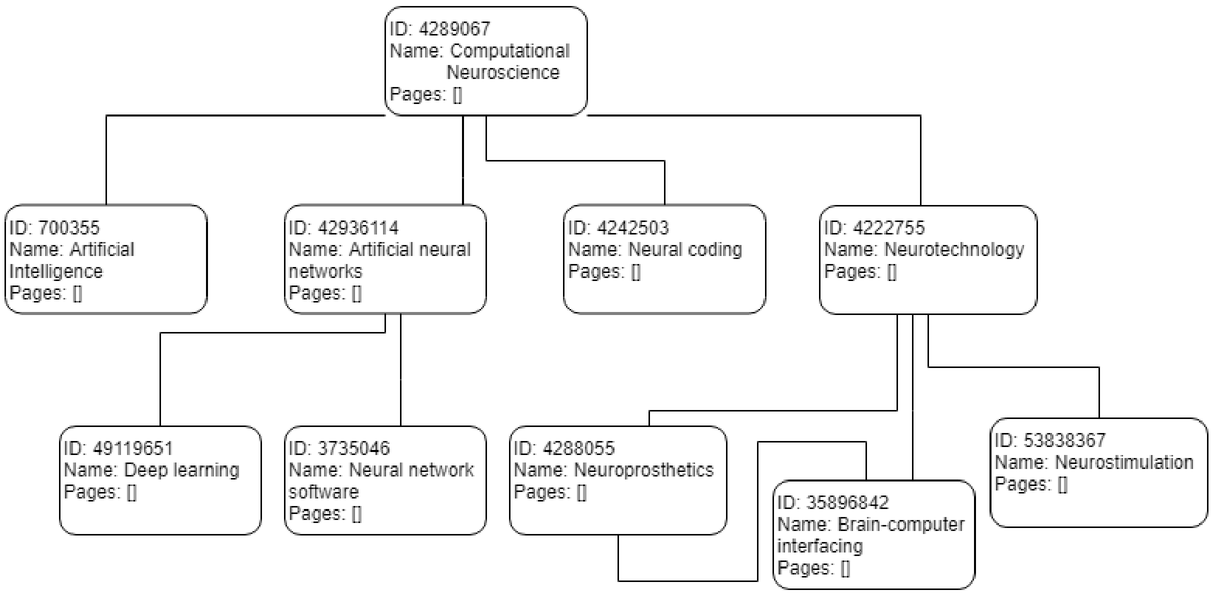


```
763530MeVitaeDictCategory:Object-oriented programming
3061615MeVitaeDictCategory:Genetic programming
12276218MeVitaeDictCategory:Software by programming language
5399121MeVitaeDictCategory:Data-structured programming languages
51277020MeVitaeDictCategory:Constraint logic programming
39817074MeVitaeDictCategory:Iteration in programming
37544678MeVitaeDictCategory:Class (computer programming)
```

I used these paths to create the relationships between the categories. Previously I have created a graph/tree-like data structure in Second Year as part of Algorithms and Data Structures. I used my prior knowledge and created a Node object to represent each category. This contained all relevant information about the category, including: name, ID, children and parents. (See Appendix 1) The Node object also contained a list for all of the pages belonging to that category which I will explain further on.

To build the tree I simply looped through the text files adding a new category as a child to its parents' node. A visual representation of a section of Wikipedia seen in appendix 2 can be seen below in Figure 3.

**Figure 3**



There were 1.4 million categories, this made building the tree very slow. The building took around 30 minutes in total. I did not like the prospect of waiting 30 minutes every time I needed to test a new function. I decided to try and optimise the building process. Through placing breakpoints in the code and recording the time taken between breakpoints I was able to see which steps took the longest time. My results showed that the longest process by far was traversing down the tree each time a new category was added and connected to an existing node. To solve this problem, I decided to place all the nodes inside a dictionary when being added. The key is its ID and the value being the Node. Now when a new category was added I could simply look up its parent in the dictionary using its ID and then connect the new category as a child. Dictionary lookup times are very fast even with 1.4 million elements. The reason for this is they do not contain duplicates and use hash values to store the data, so each key-value pair have their own index. [1] This small change reduced the tree creation time to around 10 minutes. I did not stop here though, as my supervisor suggested me to try using parallel processing to speed things up. After a brief explanation, I understood that I could create virtual threads (essentially slice up physical threads) on the computer to complete different tasks at the same time on these threads. I then split up all the text files into 10 folders and had a different thread to read each folder. To decide the number of threads, I measured the CPU usage percentage of a single thread in the visual studio debugger. I then divided 100 by this number to give me the maximum amount of virtual threads I could before bottlenecking the CPU. This dramatically improved the time down to 3 minutes and 30 seconds.

The next stage in the tree task was to now add all the associated pages from Wikipedia. Each category can have multiple pages, so this meant retrieving at least a few million elements of data. To get the page data I used the MediaWiki API to get every page and their associated category. [5] I stored them in text files with each line containing the category ID, the page ID and the page name. I multithreaded the process to speed things up. In total there was over 30 million pages and category connections with 4 million unique pages. As a page can belong to multiple categories.

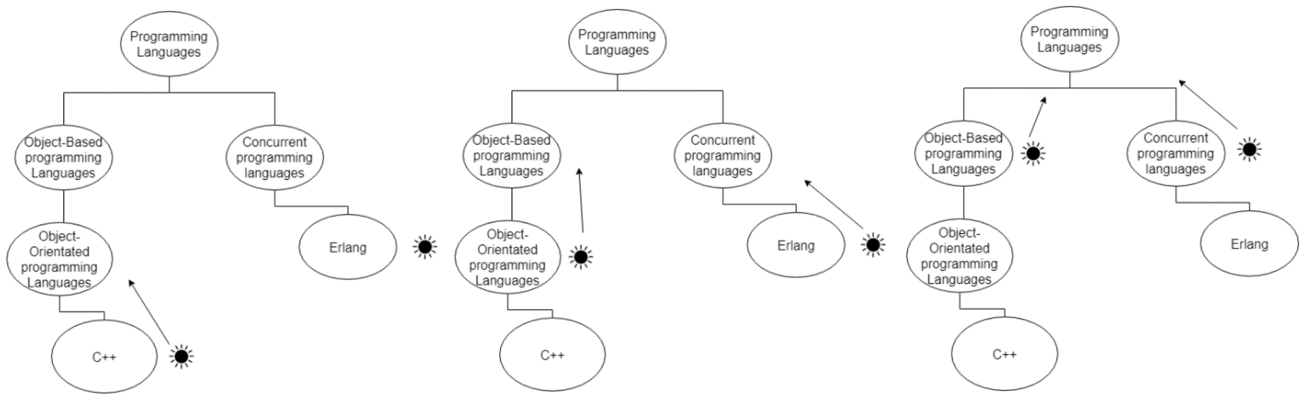
Next, I began writing the code to add the pages from the text files to the tree. As I had already learnt to use parallel processing in adding the categories I applied the same techniques here. Here though I ran into a different problem as the categories had to be built first and once done the tree was already using up 12GB of the 32GB of available RAM. When multi-threading the page adding process, I would be loading multiple of these large multiple gigabyte text files at the same time on to the RAM. This would cause the RAM usage to go above 40GB slowing the whole process down. As when there isn't enough RAM the program will supplement using hard disk memory which is incredibly slow due to the ordered structure. [2] To solve this I calculated how much RAM one thread took up, again using the visual studio debugger. I then divided the remaining amount of RAM by RAM usage of one thread to find the maximum number of threads I could use. Tree creation with pages at categories after this was at around 11 minutes.

After the tree was fully operational, I was given time to experiment. The focus of these experiments was to come up with ways in which I could use it show semantic meanings in sentences or compare sentence similarities. The first way I came up with was to classify semantics of a sentence. The algorithm for this was as follows:

1. Take a string of a sentence and split by individual words.
2. Remove stop words using a stop word list from the Python package NLTK. [6]
3. Next, for each word in the sentence search the trees category and page names for any matches.
4. Then at each category found, move up in the tree at the nodes that are below the highest found node until all positions are at the same level.
5. Now move up at all positions one at a time until a common parent is found. This common parent is the classification of the sentence.

This method proved to work for very basic short sentences that had a specific focus. For example, if the sentence was "I use Erlang and C++" the common parent classification would be "Programming languages". A flow chart of this process can be seen below in Figure 4.

**<u>Figure 4</u>**


Programming Languages
Object-Based programming Languages
Concurrent programming languages
Object-Orientated programming Languages
Erlang
C++

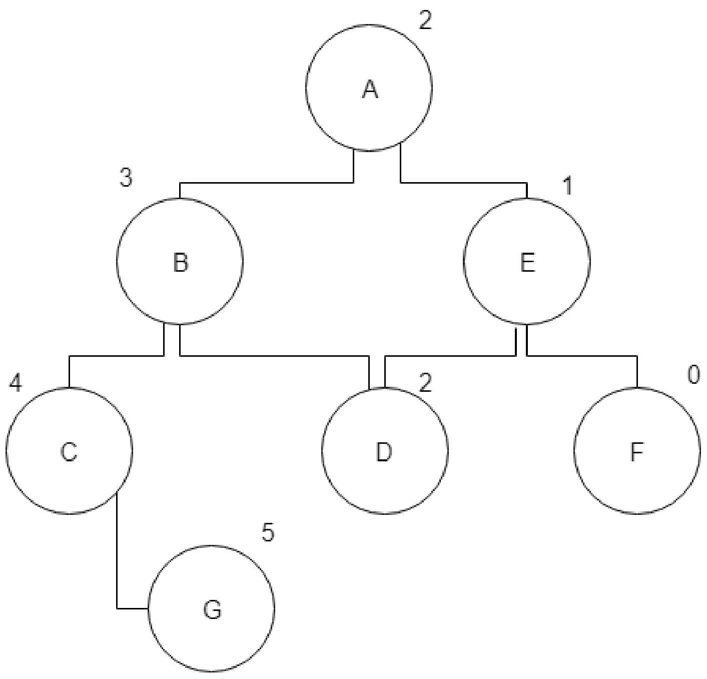
Unfortunately, if there were words in the sentence that appeared on far branches of the tree the result would usually be "Academic Disciplines" which is the very top of the tree and the first place all the tags would meet.

The next idea I had was to create word vectors from the tree for each category. These word vectors would be created by calculating the distance between a category and all other categories in the tree. A single word vector would then have 1.4 million dimensions each representing a distance. To retrieve all the distances for a category I created a simple greedy pathfinding algorithm. The algorithm explored the tree from one node/category and recorded all the distances as it traversed node to node. If a shorter path to an already found node was found the shorter path it would overwrite the old one. To calculate an overall word vector for a string of text I would find all corresponding categories from each word. I would then take the minimum from each categories word vector dimensions to create a single overall word vector. This idea proved to work very well as sentences that contained words that were related would have high similarity scores. An example below shows how word vector "F" is created from the tree shown in Figure 5:

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| F: | 2 | 3 | 4 | 2 | 1 | 0 | 5 |

Each node in Figure 5 is labelled with its distance from F.

**<u>Figure 5:</u>**



My supervisor thought these word vectors were very promising and asked me to calculate all the vectors for the categories and place them in text files to be used in the future as a lookup table. After starting this, it became apparent that it was going to be way too slow as each vector took around 2 seconds to create. 2.8 million seconds converts to 32 days and I did not have that kind of time. To solve this, I proposed that instead of using a pathfinding algorithm we should use the paths we got from Wikipedia to find the lowest common ancestor in the tree between each node. Although these paths may not always give the shortest path between nodes it will still uphold the relationships between the categories. For example, the lowest common ancestor between Java and C# might be "Object Orientated Programming" with a distance of 2. Whereas the lowest common ancestor between Java and R might be "Programming Languages" with a distance of 4. Java is more closely related to C# so a lower distance represents their close relationship.

Doing this reduced the time down to 13 days which was still too long. It was then decided that we needed to find a way to reduce to the number of dimensions for our vectors. Instead of having 1.4 million distances for 1.4 million categories have 10,000 distances. To choose these 10,000 we first filtered the 1.4 million categories to most important ones using categories that only appear on the ONET job database of skills. This reduced the categories to 190,000. The categories were then reduced further by finding the furthest apart 10 thousand from the root of the tree, so each node represented different areas of the tree.

I was now due to finish my internship, but MeVitae offered to keep me on for another month as I showed promising ideas as well as strong programming and problem-solving skills. It was decided to move ahead with my word vectors for use in CV and Job Description scoring. Now production of the product began. The first part of this involved moving all the data from the tree to SQL databases. (categories, word vectors and pages) As there are literally millions of data points uploading them all to SQL proved to be a very long process. I researched ways to speed this up and found out about SQL transactions. These work by holding a connection with the databases while data is being uploaded then committing all the data at the end. This increased upload speed dramatically as the slow part was connecting and disconnecting from the database over and over. I created three databases in total. One was all the single words present in the tree as the ID with columns that stored which pages contained the single words. The second was all the page as the ID with columns storing which categories they were in. The final database had the categories as the ID with the column as their word vector. (Appendix 3) All the ID columns (Primary Keys) were stored using a base64 hash to ensure all ID's were unique and provided fast retrieval.

Once all the databases were created coding on the API began. Visual Studio Team Services was used so that we could collaborate on the same project and commit, push and pull all our changes. At this point the work was divided into 3 areas for me, my supervisor and the CTO to work on. These were a UI for the demo and program structuring, tree-based tagging and clustering to sort the tagging. My area was tree-based tagging. My goal was to when given some text (CV or Job Description) return a list of categories that would then be sorted by the clustering. To achieve my goal, I simply would read a single word at a time and check if the single word database contained it. If it did then great, I would store the categories it was from and go to the next word. After the text was read, the categories collected got sent to clustering to be refined. (Appendix 4) Next, the remaining categories had their word vectors pulled from the database. A single vector was then created from this by taking the minimum from each dimension as explained before. This final vector then represented the text inputted. Once a vector from the CV and Job Description was found we could then use their cosine similarity to give a score. The demo was completed by the end of my last month in September, but continual work has been done since then to optimise it and give better scores. The continued work includes

- Using the tree to create training data for a Long Short-Term Memory to better filter found categories.
- Using the common parent method previously described to also better filter the found categories.
- Using phrases as well as words.

## Section 4: Knowledge learnt, and skills acquired

A fantastic skill that learnt was planning my time. I quickly learnt that if I scheduled what I was going to do for the day I got much more done. I think this is because without a clear plan you can often get lost doing aimless unimportant things. As most of my early time at the placement was open-ended, I would easily get caught up reading too many research papers and not trying to implement enough of them. Each day I dedicated my first hour to reading papers, I would then split the following day up into chunks of time dedicated to trying things I had read. This skill has followed over into university life and helps me tremendously, as I now use my time much more wisely and get distracted much less when completing tasks.

Another great skill I learnt was being proactive. At university (specifically first and second year) you can get away with not being proactive; if you do the work that has been set you can achieve a first-class degree as there is a set specification. What I quickly discovered in a start-up environment your work is never done. Even if I completed tasks for a certain week early, my supervisor would suggest multiple ways I could improve what I had already done. After hearing these suggestions a few times, I soon developed a foresight into what I could be doing extra to complete my tasks at a much higher standard. This often-included optimising and planning for errors. I plan to bring this proactive nature into my computing project, so I can go above and beyond what I set for myself.

I learnt a great deal about how to optimise programs with the of using parallel programming. As why do one task at a time when if we can do multiple at the same time if our CPU allows us too. I plan to use parallel programming as much as possible in my final year project as it involves testing multiple machine learning algorithms. I plan to run these tests simultaneously to save time.

The internship allowed me to massively improve my debugging skills. Every day I was constantly coding which means I was constantly running into bugs and therefore headaches. At university, the programs I had written before produced outputs instantly so fixing mistakes on the fly was no problem. Whereas in the internship I was dealing with such time-consuming programs. I could be waiting 10-30 minutes to see an error, then fix the error and wait another 10-30 minutes to find another one and wasted time can quickly build up. I adapted to this and started writing simple unit tests that would run instantly and test to see if my code was giving me what I expected. Unit tests along with a diligent reading of my code using breakpoints boosted my productivity massively. I think that proper debugging is one of the most important skills I learned during my internship. I have already implemented my debugging skills in my data mining coursework. In my coursework, I was to create a K nearest neighbour algorithm. The algorithm took around 10 minutes to run but this wasn't a problem as I wrote unit tests that made sure I ran into few headaches after the 10-minute run time. My new-found debugging skills will also be so beneficial to my computing project. As I will be dealing with machine learning algorithms that take up large amounts of time to run.

At MeVitae there was another intern along with me. He did not have a strong programming background so often struggled with some tasks. At university, the usual mindset is to maximise yourself and get yourself the best grades. When I first started, and I noticed was excelling past the other intern I thought it was a good thing as I thought I looked better in front of my supervisor. I watched my supervisor put in a lot of effort to get the other intern up to speed and make sure he didn't lag behind. It then clicked in me that if the other intern isn't doing well then, I'm not doing well either as we are both a part of the same company and team. I had been so used to the "every man for himself" mentality from education that I forgot I was a part of a team that should be working together. After this realisation, I would often ask if the other intern needed help translating his ideas to code and offering explanations for whatever he needed. Through helping him the work rate went up and I began to see the importance of teamwork in the work environment. Helping the other intern also cemented the ideas into my brain as teaching someone something is a great way to fully understand things yourself.

**Section 5: Self-assessment and future implications of your experience**

At the beginning of the internship, I was able to fulfil my aim of exploring the machine learning landscape. Having the opportunity to do this was incredibly beneficial as it is not often you get paid to learn. What I concluded from it is that I am deeply interested in machine learning and the problems they can solve. To the extent that I decided to take Neural Networks and Machine Learning as modules in third year. I chose these, so I can get a fundamental knowledge of these different algorithms and be able to apply and use them in the future to problems.

I was able to succeed in completing my aim of having an impact of the company. I not only wrote code that is being used in the product but I also brought original ideas, which show promising results for the company in the future.

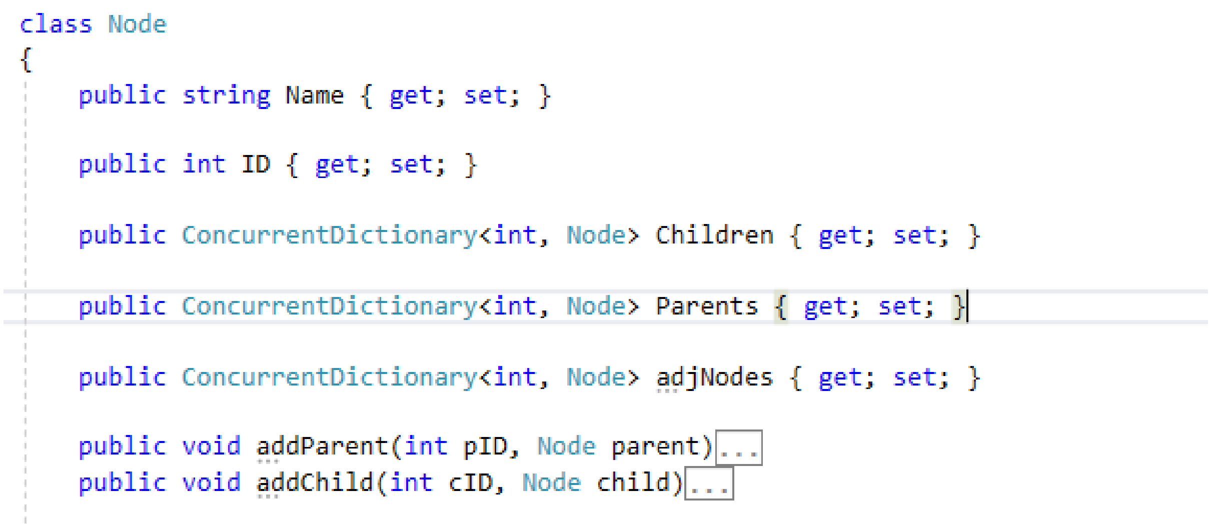
Learning how things work in a start-up was of great importance to me. Nowadays there is so much hype around start-up culture. What I learned from it is that the amount of freedom you are given is something I enjoy so much. I was often given tasks that had no set specification of how to complete them. If I met the deadline, I could tackle it in any creative manner I wanted. This control gave a great deal of satisfaction due to it being my skills that got the job done. Another thing I loved was how much impact everything I was doing had on the company. I have worked non-programming related jobs before at large companies and everything I did felt like it had no purpose or meaning. Whereas at a start-up I was seeing instant results and gratification for my efforts. At the end of the internship, I got offered a job at MeVitae and I am currently working there part-time along with my studies. I large reason for me accepting the job was the freedom and impact aspect I just spoke about.

**Section 6: Skills gained from continued work:**

As previously mentioned, I have continued to work for MeVitae. During reading week, I was given the opportunity to go with the team to Lisbon and represent MeVitae at Web Summit, the world's largest tech conference. We had a small stall on one of the conference days. I manned the stall from 8-5 along with two of my colleagues. I constantly had to do mini pitches/presentations of our company that lasted anywhere from 1 to 10 minutes of conversation. Doing this greatly increased my confidence and networking skills as I essentially used the people I spoke to as guinea pigs and quickly realised what worked and what didn't. I found that a great way to get someone interested was to ask about what their company does and then suggest ways of how our software could help them. I know that my presentations skills have improved massively from that day and will help me in my computing project presentation. A few weeks after Web Summit we were contacted by a representative of Deutsche Bank I spoke to who is interested in seeing a detailed demo of our product. This proves that my presentation skills got better as I spoke to him at the end of the day when I had lost the nerves and refined my speech.

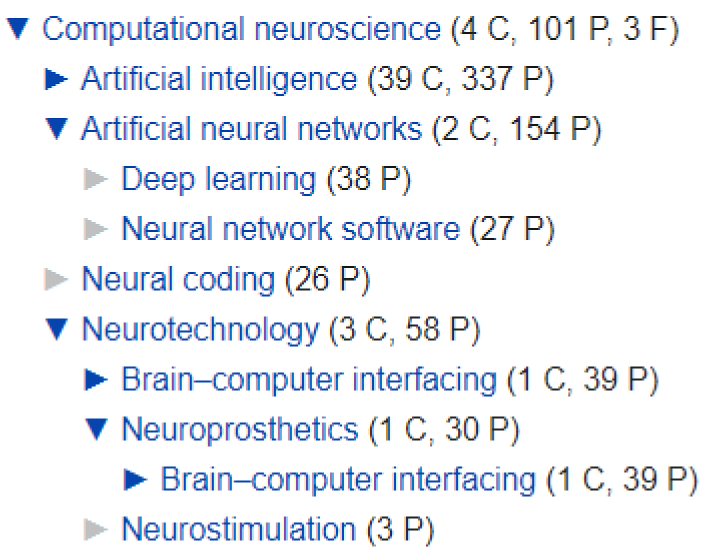
**APPENDICES:**

1. An image of the Node class from the tree



```
class Node
{
    public string Name { get; set; }

    public int ID { get; set; }

    public ConcurrentDictionary<int, Node> Children { get; set; }

    public ConcurrentDictionary<int, Node> Parents { get; set; }

    public ConcurrentDictionary<int, Node> adjNodes { get; set; }

    public void addParent(int pID, Node parent)...
    public void addChild(int cID, Node child)...
```

2. An example of categories from Wikipedia



- ▼ Computational neuroscience (4 C, 101 P, 3 F)
  - ▶ Artificial intelligence (39 C, 337 P)
  - ▼ Artificial neural networks (2 C, 154 P)
    - ▶ Deep learning (38 P)
    - ▶ Neural network software (27 P)
  - ▶ Neural coding (26 P)
  - ▼ Neurotechnology (3 C, 58 P)
    - ▶ Brain–computer interfacing (1 C, 39 P)
    - ▼ Neuroprosthetics (1 C, 30 P)
      - ▶ Brain–computer interfacing (1 C, 39 P)
    - ▶ Neurostimulation (3 P)

3. Examples of rows from each database:

Single Word Database

| ID | PAGES |
|---|---|
| Java | Java (programming language) |

Page to Category Database

| ID | Categories |
|---|---|
| Java (programming language) | Object Orientated Programming Languages |

Category to Vector Database

| ID | Vector |
|---|---|
| Object Orientated Programming Languages | 9 21 25 11 3 16 26 … |

4. Brief Explanation of Clustering:

    a. First, the 1.4 million categories were clustered using K-means and their fastText [3] (Facebook's word embeddings) word vectors.

    b. Categories that related to each other were contained in the same cluster e.g. philosophy terms were in one cluster and programming terms in another.

    c. The clustered model was saved and used as a lookup table.

    d. Each sentence from a CV or job description would often find some irrelevant categories. The program looks up which clusters all the categories belong to for a sentence.

    e. If the majority of the categories belong to a few select clusters the program will discard the categories that aren't found in these clusters.

References:

1. Microsoft. (Unknown). Dictionary Class. Available: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.7.2. Last accessed 04/12/2018.

2. Wikipedia. (2018). Paging. Available: https://en.wikipedia.org/wiki/Paging. Last accessed 9/12/2018

3. Piotr Bojanowski∗ and Edouard Grave∗ and Armand Joulin and Tomas Mikolov. (2017). Enriching Word Vectors with Subword Information. Available: https://arxiv.org/pdf/1607.04606.pdf. Last accessed 10/12/2018.

4. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. (2013). Efficient Estimation of Word Representations in Vector Space. Available: https://arxiv.org/abs/1301.3781. Last accessed 11/12/2018.

5. MediaWiki. (2018). API:Main page. Available: https://www.mediawiki.org/wiki/API:Main_page. Last accessed 12/12/2018.

6. sebleier. (2010). NLTK's list of english stopwords. Available: https://gist.github.com/sebleier/554280. Last accessed 13/12/2018