Novel use of machine learning tools for basecalling of Nanopore reads

CS39440 Major Project Report

Author: Ferenc Szpisjak ([fes9@aber.ac.uk](mailto:fes9@aber.ac.uk))

Supervisor: Dr Wayne Aubrey (waa2@aber.ac.uk)

26th April 2021

Version 1.1 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Artificial Intelligence and Robotics (GH76)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name …………………………………………

Date ……………………………………………

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name …………………………………………

Date ……………………………………………

Acknowledgements

I am grateful to…

I’d like to thank…

Abstract

Nanopore sequencing is the new generation of DNA sequencing. It has the advantage of being much smaller in size, comparable to a larger USB flash drive, while the previous generation of sequencers are comparable in size to an industrial printer, or fax machine. This reduction in size makes them ideal for carrying them in one’s pocket for instance where access to much larger machine would not be possible. However. this reduction in size currently comes at a cost, the accuracy is only about 95% for state-of-the-art sequencers, which is quite low compared to the 99.9% of the previous generation. Nanopore basecalling is a very complex process, and it is only possible thanks to Machine Learning. The low accuracy comes down to the (in)accuracy of the machine learning algorithms that are behind new generation sequencing. Several research papers have pointed out that the most error prone part is segmentation, which means grouping together raw data values that form one base.

The initial idea was to use the WEKA machine learning tool to investigate these algorithms. This is because of being an easy to use, multiplatform machine learning tool that comes with a generous amount of pre-installed classification algorithms implemented in both a GUI and Java command line version.

The first problem encountered was that there is no tool that can be used to import nanopore sequencing data into WEKA. This is an issue mainly because this is not just a simple conversion of file formats, but also a problem of the concept used to represent the data.

The second problem for the project was that basecalling proved to be a much more complex process than I originally thought it to be, and more basic machine learning algorithms like decision trees are unable to tackle it. The only tools that to date are known to provide a decent solution to basecalling are customised neural networks. For the latter reason the goal originally set for this project was not reasonable. During this project I was able to overcome the first problem, I was able to develop a tool that is suitable for using WEKA with the basecalling data. This tool could be used for reaching the original goal.

Contents

[1 Background, Analysis & Process 7](#_Toc69957826)

[1.1 Background 7](#_Toc69957827)

[1.1.1 New generation of basecalling 7](#_Toc69957828)

[1.1.2 Chiron （1） 8](#_Toc69957829)

[1.1.3 UR-Net （2） 8](#_Toc69957830)

[1.1.4 DeepNano （3） 8](#_Toc69957831)

[1.1.5 Bulkvis （4） 8](#_Toc69957832)

[1.2 Analysis 9](#_Toc69957833)

[1.3 Process 10](#_Toc69957834)

[2 Experiment Methods 11](#_Toc69957835)

[2.1.1 Custom program concept 11](#_Toc69957836)

[2.1.2 Basecall events 11](#_Toc69957837)

[2.1.3 Data Processing 13](#_Toc69957838)

[2.1.4 WEKA models 14](#_Toc69957839)

[3 Software Design, Implementation and Testing 16](#_Toc69957840)

[3.1 CDImpute class 16](#_Toc69957841)

[3.1.1 ARFF Formalities 17](#_Toc69957842)

[3.1.2 Reading in the data 17](#_Toc69957843)

[3.1.3 Processing the data 18](#_Toc69957844)

[3.2 WekaDemo class 23](#_Toc69957845)

[3.3 Chooser class 23](#_Toc69957846)

[3.4 Running the program 24](#_Toc69957847)

[4 Results and Conclusions 25](#_Toc69957848)

[5 Critical Evaluation 26](#_Toc69957849)

[6 Annotated Bibliography 27](#_Toc69957850)

[7 Appendices 28](#_Toc69957851)

[A. Third-Party Code and Libraries 29](#_Toc69957852)

[B. Ethics Submission 30](#_Toc69957853)

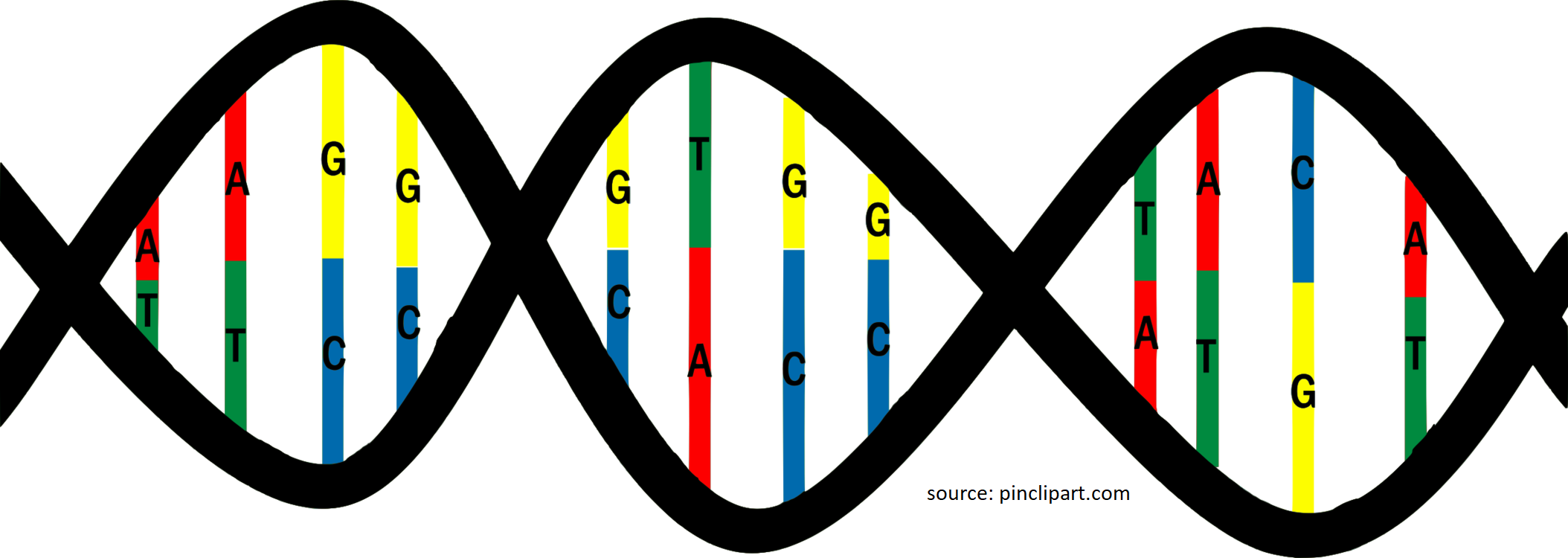
[C. Code Samples 31](#_Toc69957854)

# Background, Analysis & Process

## Background

### DNA Sequencing

DNA are made up of four types of nucleotides. The four nucleotides are Adenine, Cytosine, Guanine and Thymine. These can be found in pairs in a strand of DNA. Adenine always bonds with Thymine, and Cytosine always forms a pair with Guanine. The human DNA contains about 3 billion base pairs. DNA sequencing is determining the sequence of nucleotides in DNA.



### New generation of basecalling

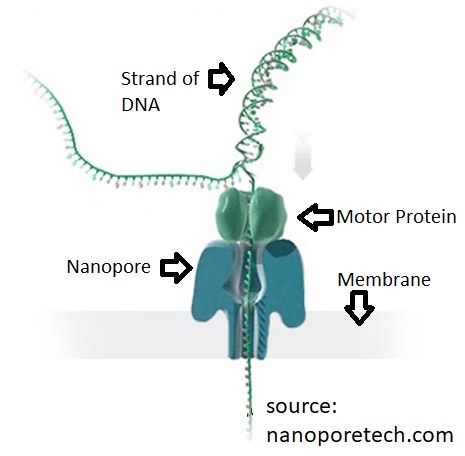
Nanopore sequencing, is the next generation technology for DNA sequencing. Before its existence, DNA sequencing methods such as Sanger’s made use of a machine that was similar in size to that of an ATM machine. While it had an accuracy of 99.9%, this older technology did not allow a sample to be sequenced in the field, the sample had to be brought back for analysis in the lab.



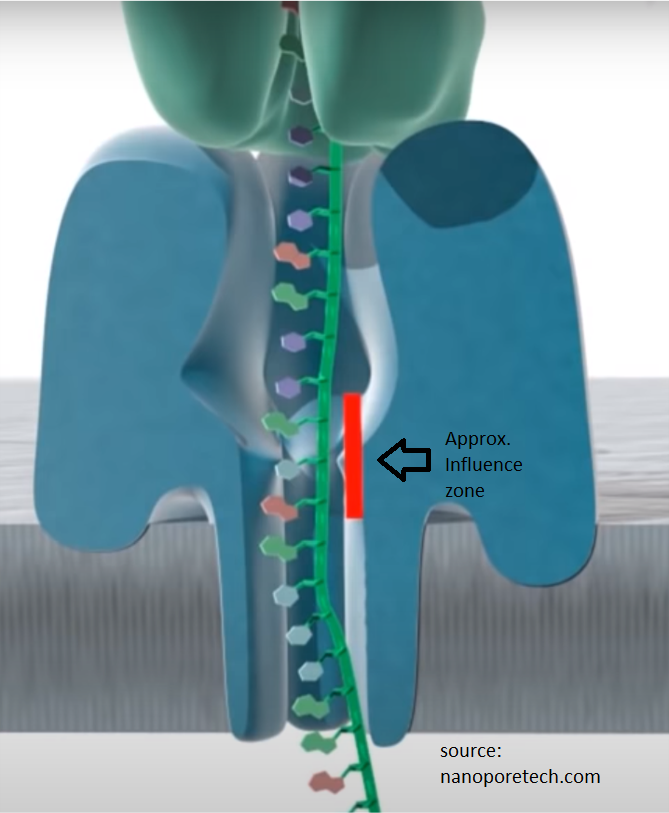
Nanopore sequencing allows the use of devices not much bigger than a USB flash drive used together with a laptop, to sequence DNA. This is made possible not only because of the nanopore technology, but the machine learning algorithms that are behind it. This technology however is not yet perfect, the accuracy rate of state-of-the-art sequencers is 95%.



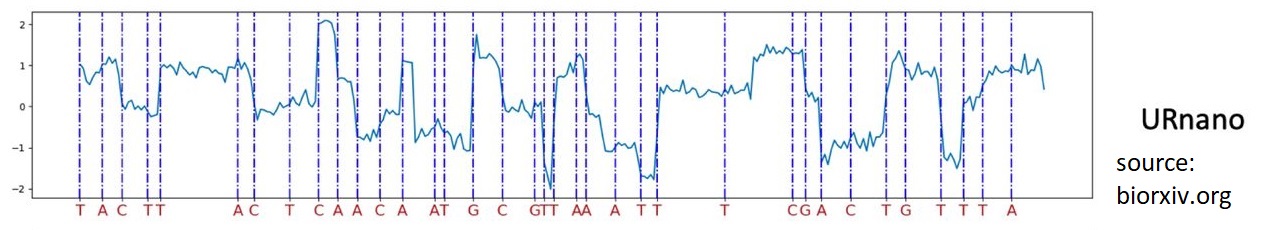
A nanopore is a pore of nanometre size, usually made out of some synthetic material such as silicone. It has a cavity in the middle, where the bases of DNA can pass through. The nanopore is sat on an electrically resistant material usually called a membrane. This membrane then has electricity pumped through it.



As bases pass through the cavity of the nanopore, they create a change in the strength of the electric current running through the membrane.



The strength of the electric current is measured at all times, and the changes in this current is what make up the raw data of nanopore sequencing. **Basecalling is the name of the act of translating the raw data into bases.** The visual representation of this data is sometimes called a squiggle, based on how it looks.



The common sequencer device for Nanopore basecalling is the MinIon (9) sequencer. It has a size of about 10x3x2 cm, and it connects to a personal computer using a regular USB 3.0 connection. It supports almost any read length ranging from short to ultra-long. Read length in this context means the number of base-pairs read in one go, creating one “piece”. It stores the raw data on the disk in “pieces”. These read pieces have overlapping parts, and these overlapping parts will be used later for assembling the entire data.

### New areas of study

Other than the aforementioned size advantages, the new generation of DNA sequencing also makes possible researching topics that would not be possible without it. Such an example is that it allows the study of Computational Metagenomics. Metagenomics is the study of genetic material recovered directly from environmental samples, such as bacteria from gut. The study of this would be otherwise tricky, if not impossible because most bacteria in the gut live in organised “communities”, and it is not possible to cultivate them separately. Nanopore sequencing makes it possible to take snapshots of these organisms in their natural environment.

### Chiron (1)

The basecaller Chiron is the end product of research into causes of the relatively high error rate. Chiron pointed out that the most error-prone part of basecalling is segmentation. The raw data are values that represent the strength of electrical current at any given timepoint. It does not clearly contain information on which values make up one base. **Segmentation in this context means to establish which groups of the raw data make up one base.** This number changes from base to base, can range from one digit to three-digit numbers. This is a non-trivial task for two main reasons. First problem is that the bases do not pass through the nanopore at a constant speed, under the same time period 5 to 15 bases can pass through. The second problem is that the raw signal is not only based on the base that is in the middle of the nanopore, but bases that come before and after the one in the middle also influence the raw signal. To attempt to improve basecalling accuracy, Chiron introduced the idea of basecalling without segmenting the data. Instead of segmentation, Chiron does basecalling directly from the raw data. This is one using customized neural networks.

### UR-Net (2)

Another approach to dealing with high segmentation error rate, is to improve segmentation. The research published by BMC bioinformatics, addressed the poor segmentation performance mentioned earlier in a different approach, by proposing a new custom network model for segmentation. This is based on the U-Net neural network, that is commonly used for image recognition. U-Net is a convolutional neural network, that is 2 dimensional, because for image segmentation a two-dimensional approach is used. However, the segmentation task for basecalling is a one-dimensional task, and for this reason, the base U-Net had to be refined. This U-Net optimised for basecalling is called UR-Net, it stands for refined U-Net.

### DeepNano (3)

Another research paper, named DeepNano investigates different problems. There are multiple tasks (such as infectious disease detection) that could be done with the help of Nanopore sequencing, however they cannot be done with the official toolkit. In order to be able to develop applications for such tasks, a good open-source basecaller is needed, as the official tools are not open-source. For this reason, they created an open-source basecaller called DeepNano. It also mentions that a new version of the MinIon sequencer is able to generate 2D reads. This was introduced by the official creators Oxford Nanopore Tech to reduce error rate. It produces 2D reads by reading both sides of strands of DNA at the same time. This opens the possibility of creating software that can catch errors in the sequencing, thus reducing error rate, because if one side of the strand is known, its pair can be predicted. Adenine(A) always bonds with Thymine(T), while Cytosine(C) always bonds with Guanine(G). This makes it easier to pinpoint potential errors, if the pairs do not match, then it is clearly an error. It uses a combination of three types of neural networks to deal with such reads. Simple recurrent neural networks are common to be used for basecalling. DeepNano introduces the use of bidirectional recurrent neural networks. This means that the network can use its most recently acquired knowledge first in backwards run, instead of just being able to start with the oldest knowledge. DeepNano also uses Gated recurrent units, which also was not commonly associated with basecalling.

### Bulkvis (4)

BulkVis is a tool to visualise basecalling data. Nanopore sequencers read data in chunks. As I discussed earlier, these chunks have overlapping reads at their start and end position, as this is used to help assembling all of them into one read. These chunks can be short(75 base-pairs) to very long(300 base-pairs). The initial goal of this research paper was to investigate their claim, that the fact that data is being split up into reads, there is actually a small amount of data loss present at the beginning and at the end of said reads, as the sequencer cannot record data briefly when changing reads. The original toolkit by ONT does offer an optional tool that captures the entire data stream coming out of the MinIon sequencer. This version also contains metadata to be able to identify the bulk information. BulkVis allows the user to visualize multiple bulk basecall files in bulk. This was developed to investigate two cases: as mentioned before they theorized that switching to a new file can result in data loss, which can impact the overall performance of basecalling. The second concerns unblocking. Unblocking is reversing the movement of the motor protein, thus ejecting bases. This is useful for two reasons: Firstly, if we require a “read until” read, when the goal is reached, the sequencing stops. The other use is removing blockages in the pore. According to this paper, unblocking can cause a change in the electric current for about 2 seconds after it occurs. But using the generic tools this cannot be investigated, because the data after unblocking is not saved.

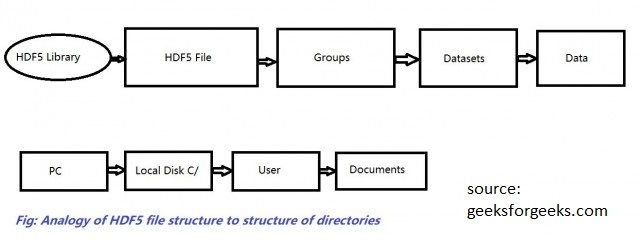
## Analysis

The researches previously mentioned defined the problem to some extent. Under time period x, sometimes 5, sometimes 15 bases pass through the nanopore. What is more, the signal strength is not only influenced by the one base just passing through, it is also influenced by bases that came before and after. To make basecalling accuracy better, either methods that basecall without segmentation, or methods that do segmentation more accurately are needed.

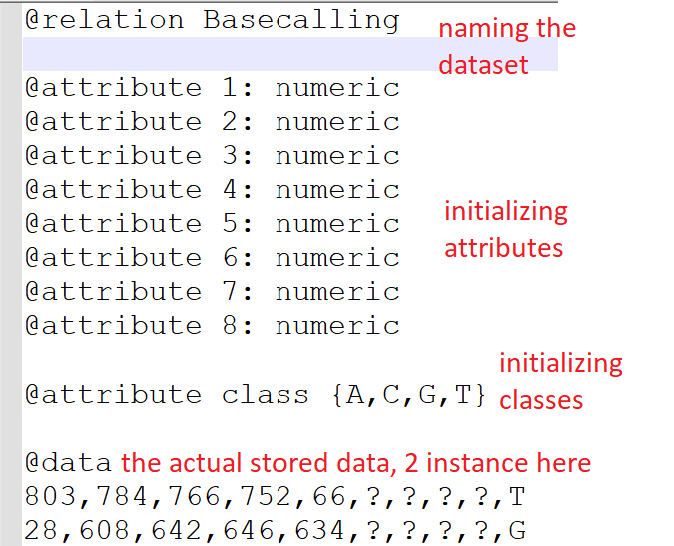
A good tool to evaluate different machine learning methods for either option, is WEKA (5). WEKA is a data mining tool developed by the University of Waikato. The abbreviation stands for Waikato Environment for Knowledge Analysis. It implements a good number of algorithms by default, including for instance decision trees, k-nearest algorithms, support vector machine models. There are also many tools and extensions available for it to further enhance its capabilities. Today’s version of this tool is based on Java, and it has an easy-access graphical interface, as well as a Java command line version available. It has tools for multiple uses, including data pre-processing, classification, feature selection, and data visualization. It also makes comparing different classification algorithms easy as there is a method (WEKA Experimenter) to run multiple classifiers for comparison on the same dataset for a (relatively) fair comparison.

The first challenge is to be able to import basecalled bases and raw signal data pairs produced by the albacore basecaller, into WEKA. They have to be imported in pairs for the training phase, as this is supervised learning. This is not straightforward because to store basecalling data a format called fast5 is commonly used that is based on the HDF5 format, while WEKA supports its own, ARFF (Attribute-Relation File Format) file format.

The fast5 format is based on the HDF5 file architecture. The HDF5 format uses a file-directory like structure to store information.



Below is how the ARFF format looks like. One file is called a relation. The first line sets the name of the relation. After that all attributes have to be given. It is important to note that all data instances must have all attributes. Then all the possible classes have to be initialised. Lastly comes the actual data, in the format of instances. An instance consists of all the attributes separated by a comma, and at the end the class this instance belongs to, also separated by a comma.



Fast5 files cannot be viewed with common text viewers/editors. They require specialised tools that can export them as commonly used file extensions.

Once the data is in a much more accessible format, it will be much easier to devise steps for the data to be transformed into an ARFF format for WEKA.

After the data can be loaded into WEKA, different approaches and tools can be evaluated. What approach means in this context, is the concept of representing this type of biological data in the WEKA format. ARFF data is made up of instances, each instance having their own attributes and a class. It has to be investigated how the bases translate to classes. Also, all instances in ARFF have to have the same number of attributes. This however is not the case for basecall data, because different reads have different length, partly because of the varying speed of bases. At this point, it is not known for me, how different are the quantity of attributes. This is to be investigated, together with a method that is able to import data with a varying number of attributes.

## Process

I made an initial model of the problem based on my understanding from reading the research papers. Then I tried implementing a program that would make my work possible.

To evaluate how my concept does, I compared the 10-fold cross-validation classification results to the mathematical probability of guessing correctly. For example, for models with 4 classes in total, with an accuracy of about 26%, I considered not good enough, as purely guessing 4 classes would in theory mean a 25% accuracy, at least according to the law of large numbers. While this might not seem like a good idea at first, I did this because the project is about evaluating how well algorithms do, at this point in time I can’t know how well an algorithm should do. This is why I used this mathematical concept for evaluation of my work.

The program did not work because there were flaws in the concept, so I went back to read even more research papers, and then rethink the concept. I did this until I seemed to be getting a reasonable result. This result that I deemed reasonable ended up being 1.7% accuracy for a total of 1024 classes. Guessing one class from 1024 would be about 0.1% accurate, and 1.7% is about 17 times higher, and it seemed to stay between 1.5% and 1.7% using multiple different datasets, and also using a lot more datasets for training, so this seemed to be doing okay considering I knew nothing about the suitability of the algorithms I was using.

# Experiment Methods

### Custom program concept

A custom tool had to be created to be able to use Albacore basecall data with WEKA. Upon close inspection of the output data of the Albacore basecaller, multiple patterns can be found. Firstly, the process of basecalling is described in detail, each line of data belonging to 5 raw signal values, so 5-time units. Each line here has the same format:

I have 3762 files to use for this project. Each file contains one sequencing run. All together they are about 562 megabytes, in fast5 format. Each file contains on average 9000 basecall events. However, events where a new base is found, are only about 1500 per run. So about 7500 events have moves of zero, the meaning of this will be described later. This totals 3762\*1500=5643000 bases. How many instances this means in machine learning terms depends highly on the model used for representing this data, but even the best-case scenario is one fifth of the number of bases, which is 1128600, so just over a million. This means that if I am to include all files, initially I do not think that non-linear classification algorithms, anything with at least quadratic time complexity can be used. Depending on the model, I will evaluate multiple cases later on.

### Basecall events

This is how one basecall event is described in the Albacore basecaller:



The first float value is the mean value of the raw signals of the period

The second integer value shows the start time point of the line

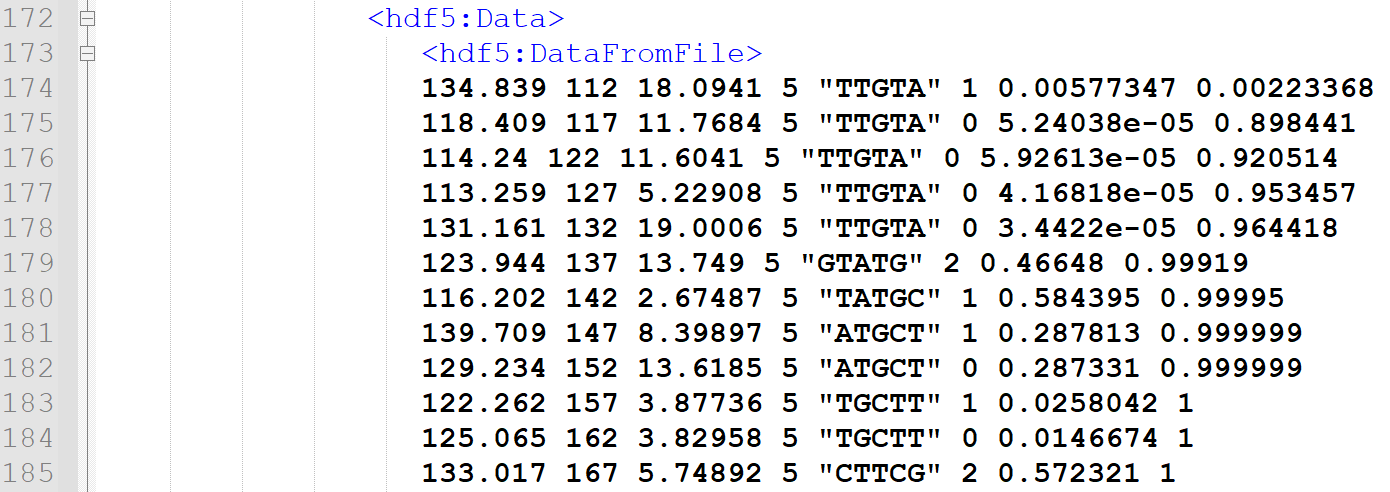
The third float is the standard deviation, so how much the mean differs from the average mean

The fourth integer is the length, this says how many raw signals correspond to the event, this is 5 in all cases here

The fifth is a string, called the model state. This shows the last 5 called bases.

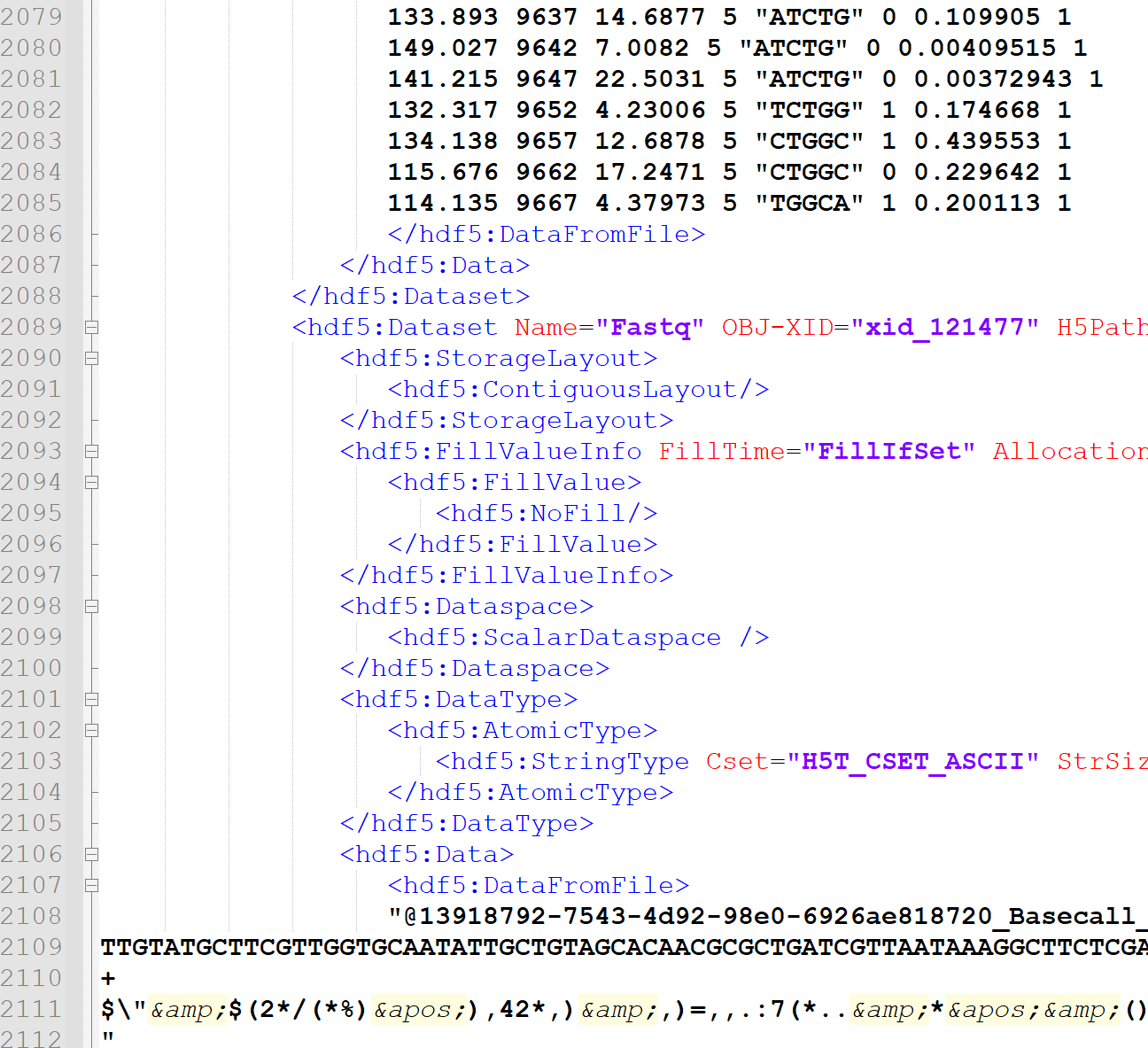
The sixth integer is the move, which can be 0 or 1 or 2. This tells how many bases were added to the model since the previous event.

These descriptions of the basecall events always start at line 174 of the file. This kind of information extraction is safe to use as the source code of this type of file was not meant for viewing, and there are no line breaks because of long lines at all, all line breaks in the file are dictated by its format.



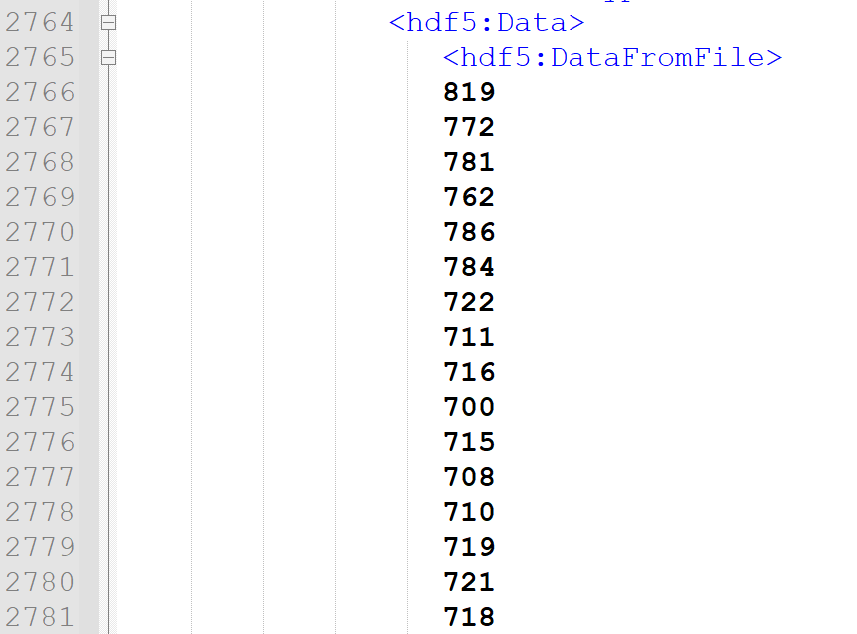
StartOfEvents 1

After the last line of the events, always exactly 24 lines below is the long string of the complete basecalled bases. This is about 2000 characters, but stored in one line. The events always start in a number, and when the list of events end, this is marked by a xml style end of element. Because different files contain a different number of events, the line number of the end of events differ from file to file, but the string of bases is always 24 lines below.



endOfEventsAndBasecalls 1

Then always exactly 657 lines below is where the raw signal data are stored, one value in one line.



These are all very useful information to be able to extract the information, allowing the transformation to WEKA’s data type, arff.

### Data Processing

Some adjustments are to be made to this data: This picture: StartOfEvents 1 shows that the events start describing from time point 112 onwards. This means that in the raw data, the first 111 value are to be deleted, as they do not belong to any of the bases.

Picture endOfEventsAndBasecalls 1 shows that the last event starting timepoint is 9667, so the last time looked at for basecalling is 9667+5=9672. This means that if these are any raw data starting from 9673 timepoint, they will have to be deleted as well, because again they do not belong to any of our bases. To find out if there are data after the events end, I did the following: In every file events start at line 174, and in this particular file, it ends at line 2085. There are 2085-174=1911 event lines. Each lines describes 5 raw signal data, so 5 time point. This means there are 1911\*5=9555 raw signal values that the basecalling describes. For raw signals, they start at line 2766 and finish at line 12441. So there are 12441-2766=9675 number of raw signal data. The first 113 needs to be deleted as described above, There are 9675-113=9562 values now. However basecalling described only 9555 as earlier mentioned, so the answer is yes, there is still data after the events stop. These have to be deleted as well as again they do not belong to any bases here. 9562-9555=7, so the last 7 raw signal is to be deleted.

Making a WEKA-style dataset out of this is possible by making a program that consists of three steps: first extracting the information, then processing it, and then outputting it in the required format. Read-in: Using Java’s Scanner class, a jump to line 174 is made. The first float is ignored, and the second integer is saved, as this marks the time where basecalling events start, as described above. This is important because all of the raw data before this point has to be deleted, because the basecalling events do not describe them, this deems them useless for machine learning training purposes. Then from each line the number of moves of the DNA bases are saved in a list. These are to be able to tell how many of the raw signal data is used to determinate each base, for as mentioned earlier, this is not constant. Example: if starting from 227 there is 0 move, then the next event gets checked, starting from 232 it is also 0 so moving on to the next, starting from 237 there is 1 move, an ‘A’ base was added in the model state, this means that for this ‘A’ base belongs the raw data from time 227 until time 236. It is also possible to have a move of 2 at once, as mentioned earlier, this means that from time X until time Y belongs to not one but two bases.

The end of the lines of events are marked by the fact that the line no longer begins with a float. When this happens, the Scanner is scroller down 23 lines, where the complete result of the basecalling, the bases can be found in one line. This is saved.

After this, 657 lines below is the beginning of the raw data, so the Scanner is scrolled there. Each line here contains one integer value. All of these are saved in a list. The end of these lines is marked by the line not beginning in an integer anymore.

As for processing the data, it was saved that the data starts at time 112, so the first 111 instances of raw data are deleted here. It is also saved that the last event starts at 9667, because events don’t always cover the raw signals until the end, and any raw signal data after the end of events have to be deleted, because the base calling events do not cover these. In this case as they cover 5 units of time, any data after 9671 is deleted. For the string of bases, the first 5 bases are deleted. This is because the data starts with the model state being full with 5 bases, these were formed from the raw data before the start of the events in the data, and as previously described, their raw data counterparts were also deleted.

I have experimented with representing the data in WEKA in multiple ways.

### WEKA models

The raw base data belonging to each instance of bases, are considered to be attributes. The bases are considered to be the classes.

#### First approach

My first program builds a model that considers 4 classes in total – one class for each base A,C,G,T. As WEKA requires all instances to have the same number of attributes, I used the method that saves the maximum number of attributes among all instances, and for every occurrence where the instance has less attributes, they are being filled with questions marks, representing missing data. There are classification algorithms that just ignore missing data, and there are some that do not do well with missing data. This is something that has to be taken into consideration. At this point what I think will be a bigger problem is that feature-selection wise, the emphasis will naturally shift towards the attributes that more instances have, the first X number of attributes. This will in my opinion cause the classifiers to do much more poorly on instances that are long reads, so the ones that have a high number of attributes.

I started experimenting with an initial amount of 300 files, this is about 270000 instances. This has an average accuracy of about 26%, which considering the fact that there are 4 classes, is purely the percent at which we could predict classes if we were to start blindly guessing. I tried the accuracy with just 1 file, and it is only on average half percent lower. This tells me that this model approach is not good enough, because even if this was to work, it would need too much data.

#### Second Approach

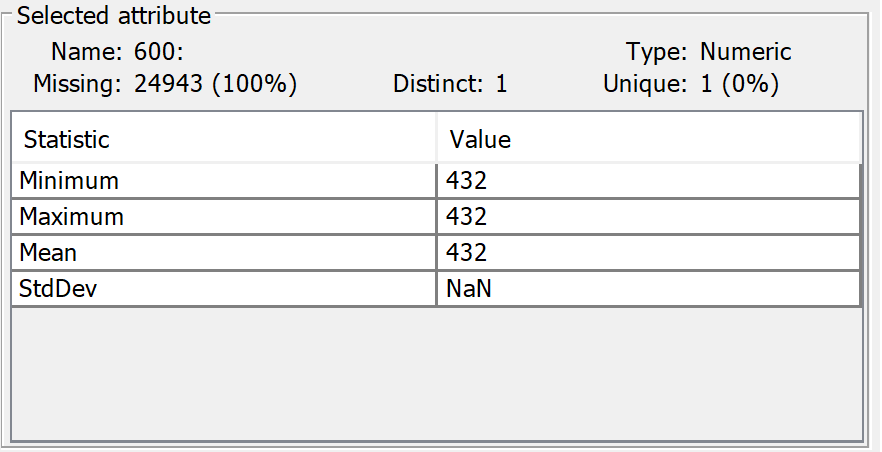
The second concept that I have tested is to have classes made up of 5 bases. So any permutation of length 5 of the bases A,C,G,T. This results in a number of classes equal to 4 on the power 5, which is equal to 2 on the power of 10, which is 1024. This is a lot of classes, and to be able to see this working, I predicted that an enormous amount of training data will be required. The concept of raw data as attributes is the same, however as now they of length equal to 5 bases instead of just one, the average number of attributes should be about 5 times higher per instance, however the number of instances for the same amount of data should be lineally reduced as well. In theory, this should result in reduced training time compared to classes of one base, as typically a higher number of instances have a much higher effect on training time then the number of attributes each instance has.

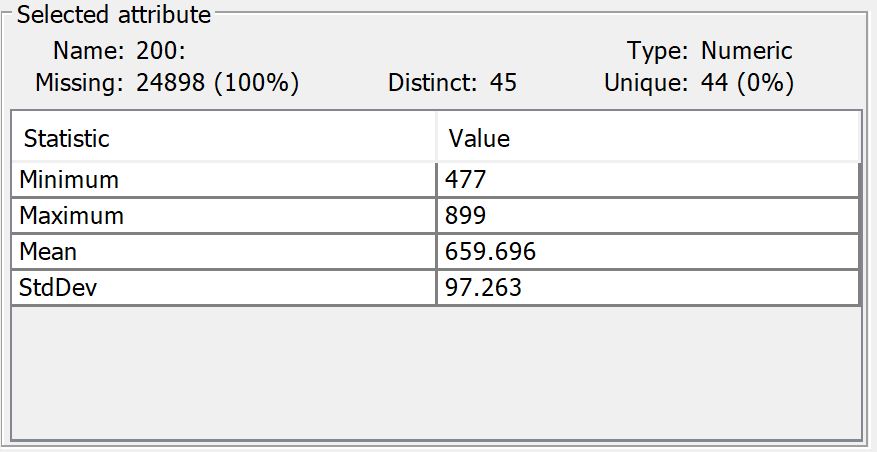
#### Third approach

Models in the research papers I read were mostly using k-mers. This in a way is a hybrid approach of my previous two approaches, so this is what I went with for my third concept. Based on k-mers I have tested representing the data in 5-mers. This is a sliding-window concept. For instance, the seven bases ACGTACG with moves of one would be represented as three 5-mers: ACGTA, CGTAC, GTACG. If there is a move of two, this would mean only two 5-mers: ACGTA, GTACG. If moves of one and two are mixed, the length of the bases of the previous example have to be extended by one, this looks like the following example: ACGTACGT – ACGTA – first two moves: GTACG – then one: TACGT. It is also possible to have the move of first, and the move of two second – ACGTA – CGTAC – TACGT.

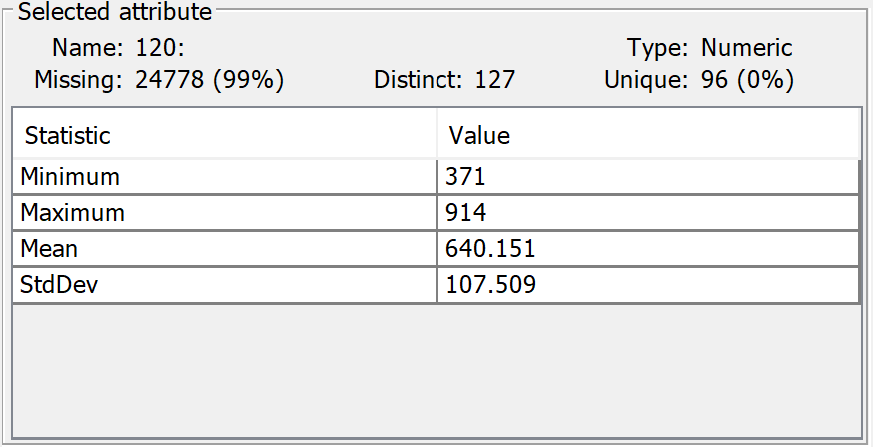
This method is what most researches I have looked at used, however they the length of k-mers varies, DeepNano uses 6-mers for instance. This method is a refinement of my previous attempt. However, this method will result in a much larger number of instances, something that will significantly increase training time. The reason for the growth of the number of instances is that the number is equal to the number of moves that are not 0(So moves that are 1 or 2). This takes the number towards the first model, while the previous instance had a number of instances equal to the number of events divided by five.

Taking 200 files into the program, I loaded the output file into WEKA. First, I had to see under how many time periods a 5-mer is created. The program run allowed reads to be as long as 600 time periods, however it is clearly visible that these are some rare outliers. This file contains 24943 instances, and but when looking at how many instances have at most 200 this number is 24898.





This means that out of about 25000 instances only about 100 has more than 200 attributes. Looking at 100, about 500 has more than 100. Naturally I would like to work without data loss, but the nature of WEKA is that every instance has to have the same number of attributes. This means that I am to have 1 instance with 600 attributes, and 24999 instances with just 100, for every one of said 24999 I would have to mark the remaining 500 attributes as missing by filling it with a question mark. This significantly surges the size of the data. To significantly reduce my training time, I had to find a sweet spot, loosing as little data as possible while keeping training time reasonable. I have made this spot 120 for testing.



Filtering out any instances created over a time period of 120, and then running this in WEKA using a basic neural network with default settings, using a 10-layer cross-validation resulted in 1.554% accuracy. Instances from 200 files are nowhere near enough training data, however considering that there are 1024 classes, mathematically speaking a generic guessing result would be 1/1024≈0.0009766%. A result slightly over one and a half percent, with this amount of training data used, seem to suggest that the model works, but to what extent, it has to be further evaluated.

# Software Design, Implementation and Testing

Because there is no tool currently to my knowledge that allows for importing fast5 files to WEKA, I decided to take this project towards creating a tool that acts as a bridge for this gap. A GUI tool is always good, but it is much simpler for now to create a tool that can be executed through the command line. Because tools that allow the extraction of information from fast5 files are widely available on Linux, but are not common on Windows, my tool currently is only for the Linux command line. It allows the user to train basic WEKA algorithms for basecalling and evaluate their performance. This tool is not mainly aimed towards these basic classification algorithms, and if I were to continue this project outside of the time limit, the next step would be to integrate Eclipse’s DeepLearning4J (6) into the algorithms my tool works with. For now, it can only use the base WEKA classification algorithms. My program makes use of the Linux command line tool called h5dump, my java code, and the java command line version of WEKA. My java program will be run by a bash script.

I will only describe the final model’s program, as I made multiple versions that ended up not working, and that’s how I went back to rethink my model.

## CDImpute class

The class that I have built in this program is called CDImpute. This is what I will be describing first.

### ARFF Formalities

First is the run function, this is the function that has to be run in order to execute what this class has to offer. This run function when called from the main function has to be passed a directory where all the fast5 files are. This is the directory where my program is located. The idea is that in order to execute the bash script, one has to cd to the folder where the program is. The current directory then will be passed to the main function as a system argument from the bash script. The using Java’s directory listing the program searches for all the files in that directory, and puts all the filenames in an arraylist. Then it calls the next function, imp.

Imp function builds a file for WEKA use based on my third approach(2.1.4.3).

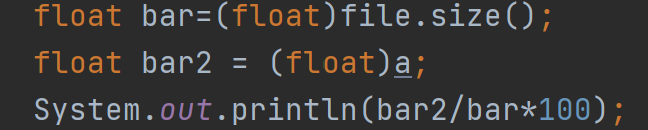
First it creates the output file, and writes the necessary text for the attribute relation file format. It also sets all possible classes, so all possible permutations of length 5 using the 4 DNA bases, (repetitiveness is allowed) which is 4 on the power of 5, which is 1024.

### Reading in the data

From here, the rest of this function will be inside a loop, so that the rest of my code here has to be executed for each file in the array list of file names created earlier.

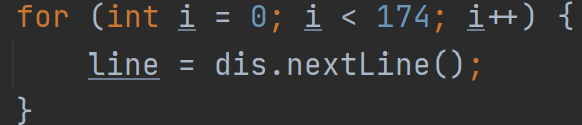


Next is a printing a report on the process at the beginning of each file, so it prints how much of the work(how much of the total number of files are converted) is done in percentages.



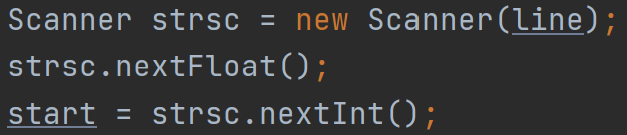
Then I create two integers, one that says start, and one that says end. These will store start and end time point of the events in the current file. ArrayList steps will contain the the complete basecall string. ArrayList signals will contain all of the signals from the file. ArrayList save is an array that will be temporary placeholder that is required for the functioning of the program. More on this later.

A java scanner is created to read in text from the file passed in the beginning. The scanner will jump over the first 173 lines to end up at line 174 where the basecalling event descriptions begin.



I am interested in saving the number of steps (0 or 1 or 2) from each event. Also, from the first event the timepoint, as this is the timepoint where basecalling events start. And also from the last event line the timepoint, as this is where description of basecalling events end.

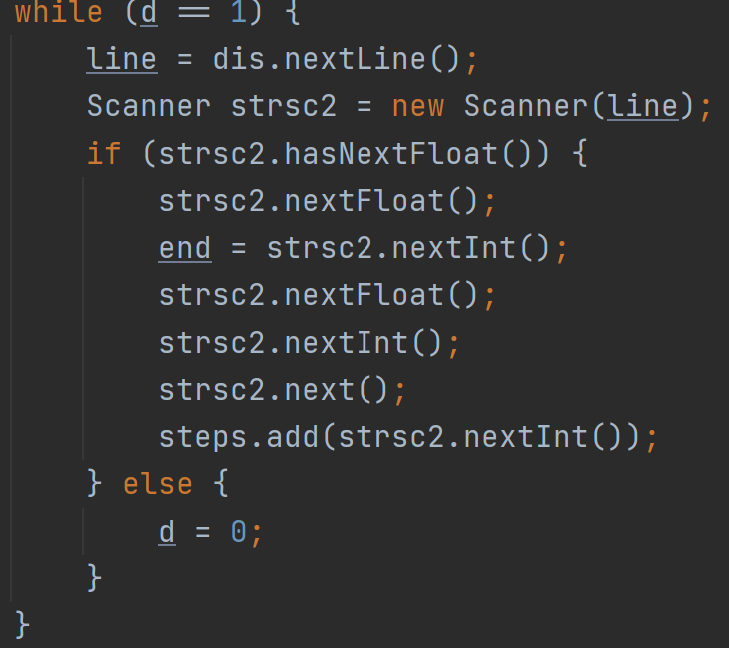
To go ahead in the code, it is now going to skip over the first float, and then save the timepoint of start to the integer I created earlier for this purpose.



It is also going to save the number of steps the model has moved, this is the 6th data. To do this, a new scanner is created, so that it begins from the beginning of the first event again.

Then the code enters a loop that saves just the number of moves from every line, until it realises that the line it is currently on is not a basecall event anymore. This can be realised because the lines would no longer start in a float, rather in text.

Also at every line it sets the current time point as end time point. Then in the following line it will be overwritten, so in the end the end value will be the time value of the last event line.



Then it skips down 23 lines, where the entire basecalled bases are found in one string. It saves it into the string basecalls.

Then it skips 657 lines below, where the first raw signal data start. These are just one integer in one line. So it keeps adding each integer into the signals array list, until it reaches a point where the line does not start with an integer anymore, thus there is no more raw signal data.

### Processing the data

After reading in the data, comes processing said data. There is data in there that that not fit the model, and it also has to be transformed into the right data format.

#### Filtering out unwanted data

First step here is to recognize that the **first line** in the events already has **5 bases** and **a move** filled out.

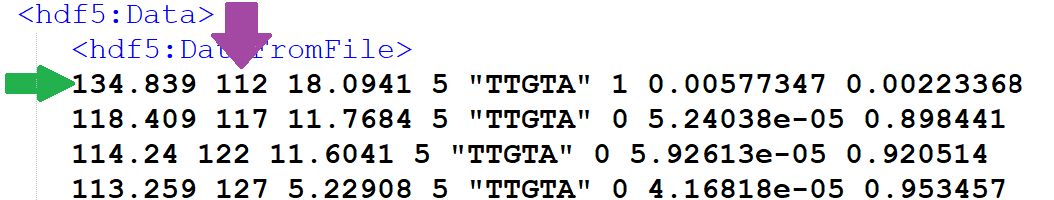


We do not know anything about how these were created so they have to be deleted.

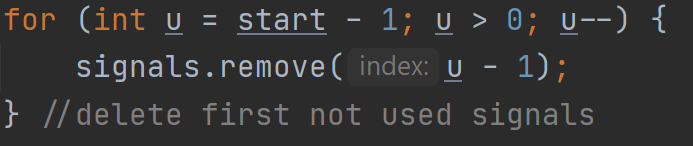
So the first 5 bases and the first move is deleted.

As mentioned earlier, basecall events usually do not cover all of the raw data. Some data in the beginning and in the end are not covered, so these raw signals that are not covered by events have to be found, and deleted.

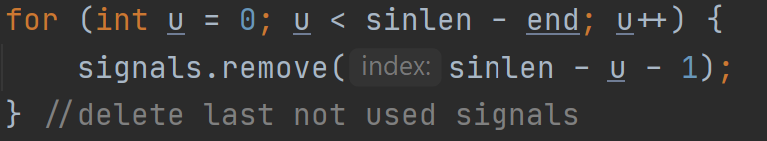
Here for example the **first basecall event starts at raw data event number 112**. This means we will have no base pairs for raw data from time period 1 to 111.



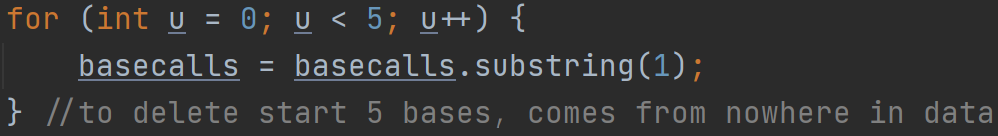
The deletion of not covered raw signal data is done by having the length of all raw signal data as the integer sinlen, start is where the events start, and end where the events end. So from the beginning of the raw signal data, I delete start-1 elements, as these are the ones not covered.



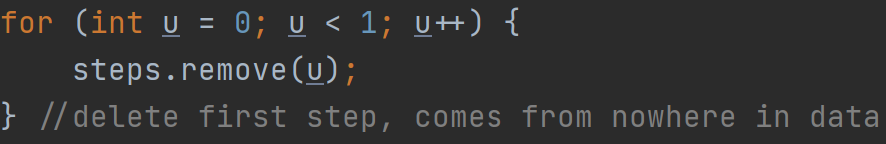
Then going backwards from the last item, I have to delete sinlen-(end-start) number of items, because I this is how many more not covered raw items are at the end. Sinlen again is the current length of the raw data array, it is important to get updated to after the deletion of the first start-1 items. And end-start is how many elements are covered by the events.



The first 5 bases are simply deleted from the String of bases.



So is the first move removed



#### Grouping data for ARFF format

Then comes a while loop which is set to go on forever, but it has a break condition that comes active when all the data has been used.

How the contents of this huge loop works, is that each integer in the moves array gets checked one by one. There are different steps to take for each of their 3 possible values. It was earlier described that one basecall event corresponds to 5 raw signal data.

##### Filling in the initial k-mer

Firstly, the program works by building a k-mer model, and outputting each state to the outputfile. At the beginning, everything starts empty, so the program cannot write anything to the output file until it is initially filled up with data.

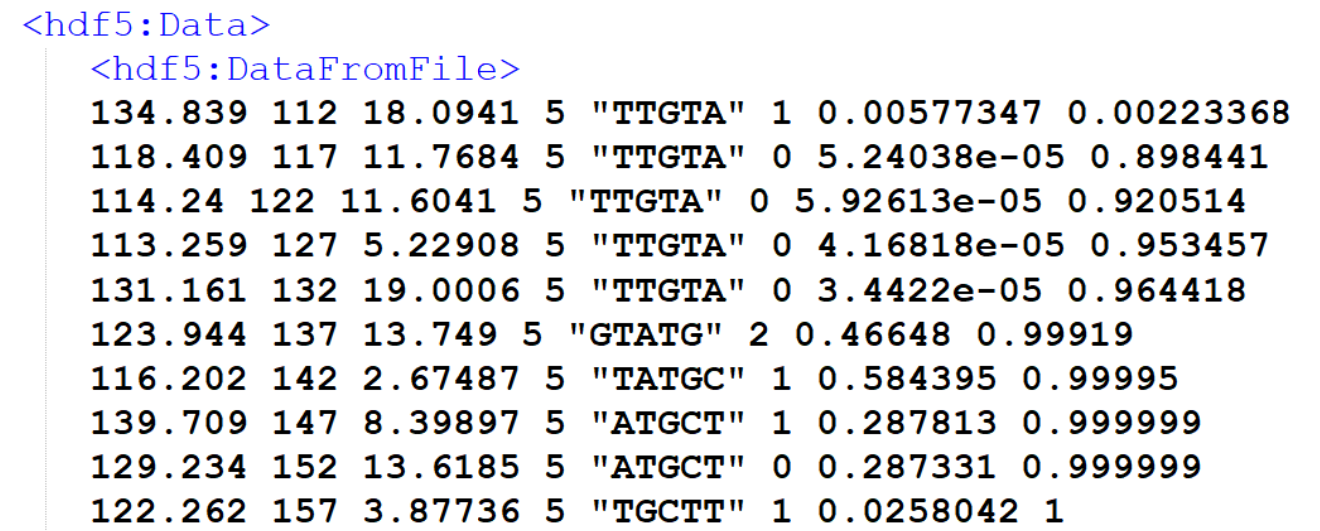
If the first move is 0, then the first 5 raw signal data gets stored into an array. Then integer x gets incremented by one, this is the integer that tells the program which element to check in the steps array. So it moves on to the next step value. At the same time, c gets incremented by 5, as c is the value that tells us which 5 raw signal data to handle. If the next move is zero again, this is what will happen again, save the 5 raw signal data in the array on top of the ones that were previously saved, and increment move index by one, and raw signal index by 5.

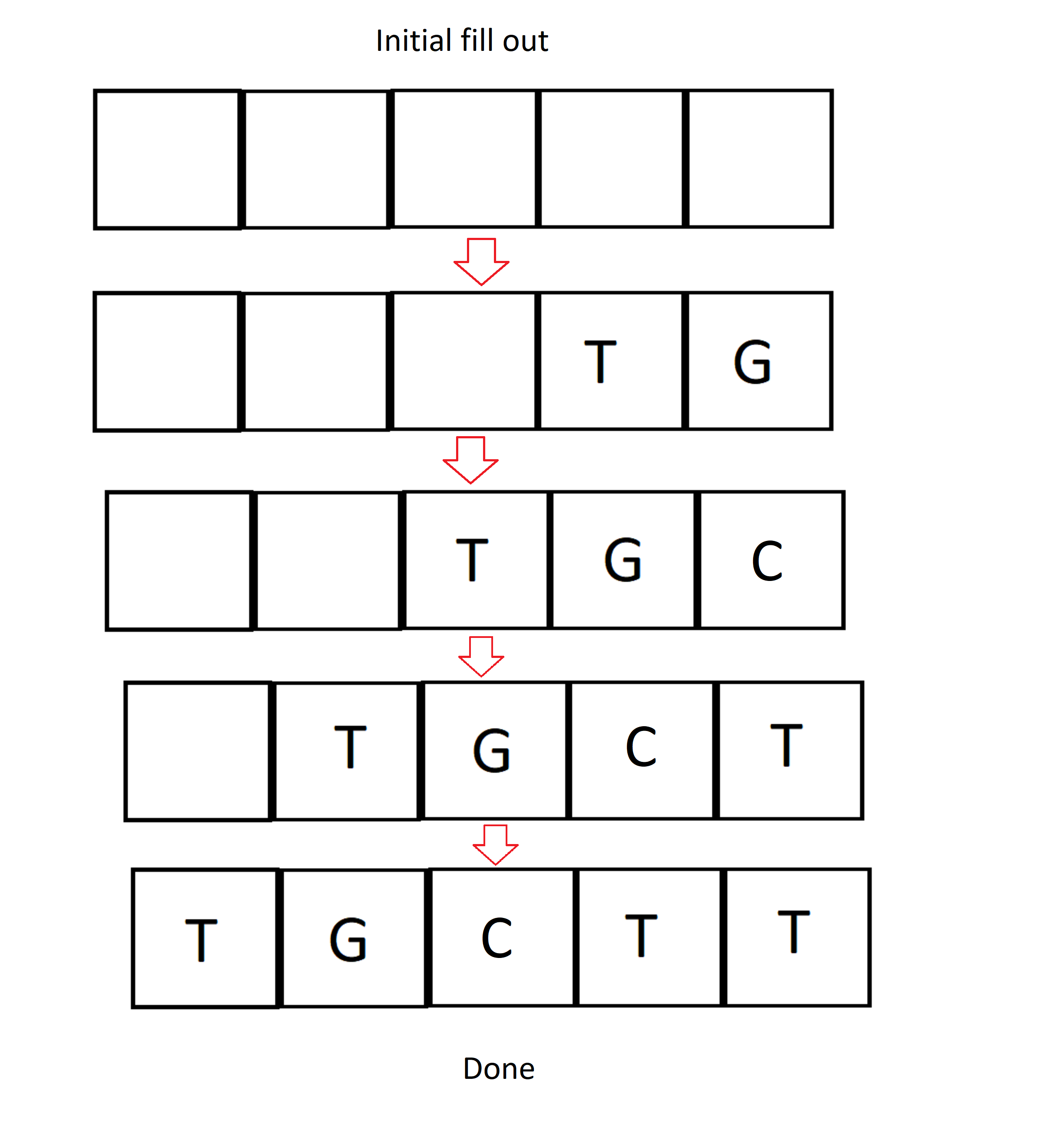
If the move is 1, then the current raw signal data is added to the array as before, the difference is that the first base gets written into our current model. Then the correct increments are made for indexes, +1 for moves, +5 for signal, +1 for bases.

If the move is 2, then the only difference from those that happened in move 1 is that 2 bases get written into our model instead of one. And then as a result the bases index will be incremented by 2 instead of 1.

These moves go on until our initial model gets filled with 5 bases.

Below are some events, and below the events is a representation of how this would fill the sliding windows out.





The moment our model has 5 bases filled up, the model is written out in the output file along with all the raw data that has been save up so far. Then the program empties its save array. Now, in the beginning the model has zero elements, it gets loaded up to 5, and from there on out it always consists of 5 elements. If by any chance that were to occur that the model is in state 4(has 4 out 5 bases inside), and a move of 2 comes, then the program treats this as two separate 1 moves, without deleting the save array after the first move of 1, to make sure that there is no data loss. So what happens is the first base of the 2 move gets filled into the model, and the five bases are written out along with all the saved signals. Then in a sliding windows concept, the first base in our model gets deleted, and the other base from the move of 2 gets added to the end, and this model is written to the file along with all saved signals. Then the signal array is emptied, and the move is incremented by 1, the signals by 5, and the bases by 2.

##### Starting to work sliding window style

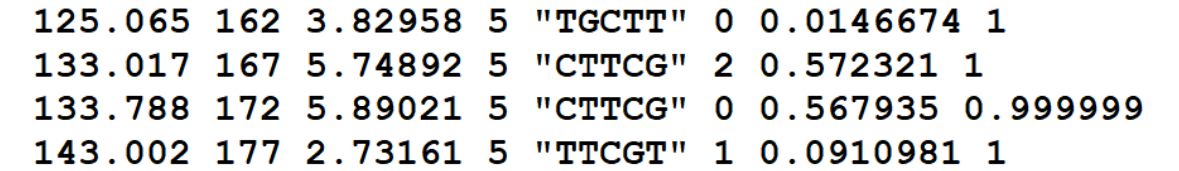
Now that the initial filling up of the model is done and it will always stay at length 5, from here on out, the following will happen:

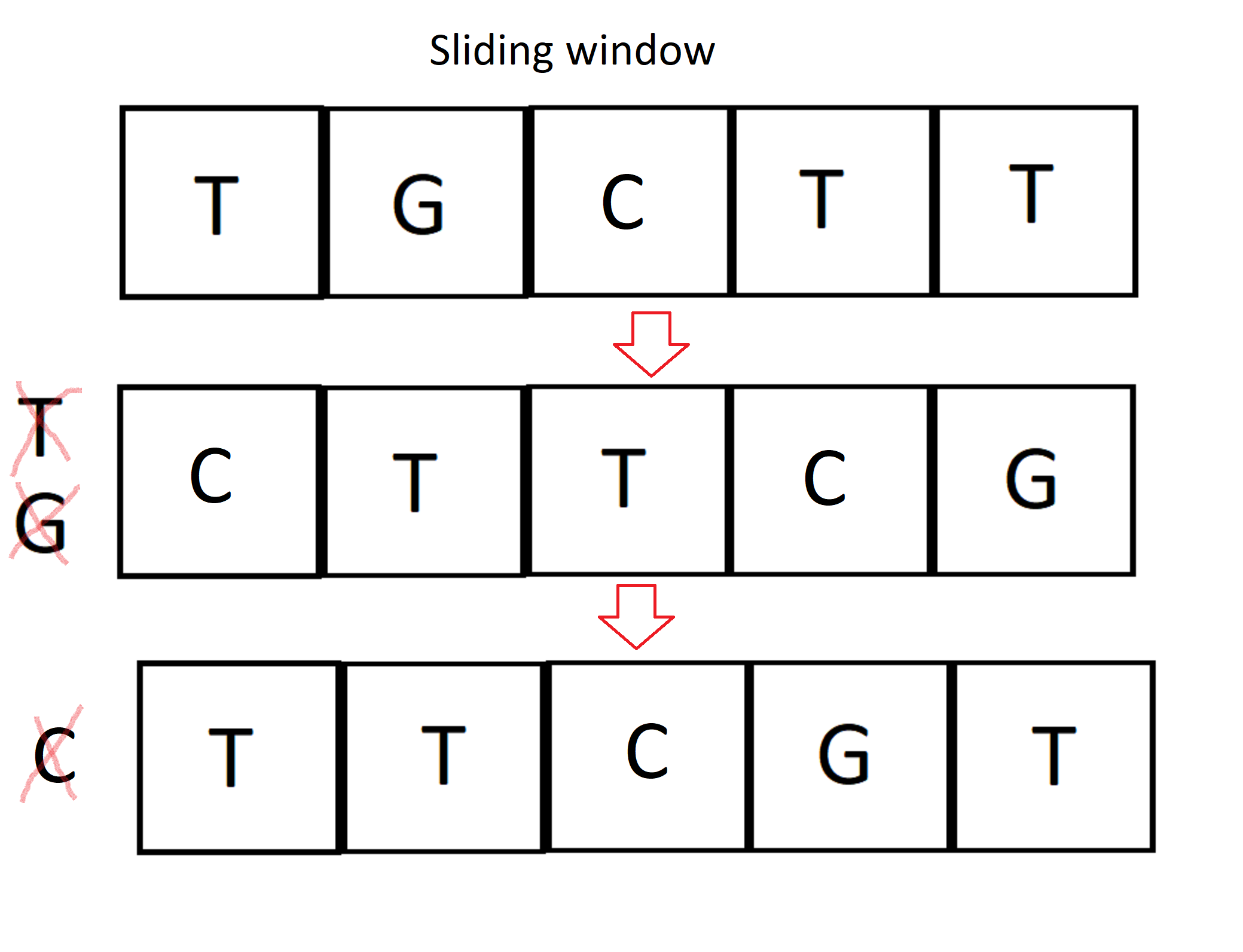
If the move is 0, save the 5 signals to array, increment move by one and signals by 5.

If the move is 1, delete first base in model, insert new base to last place in model, add 5 signals to signal array, write signal array along with model to file, delete signal array, increment signal index by 5, move index by 1, basecall index by 1.

If the move is 2, delete first and second bases in model, add 2 new bases to last 2 places. Save 5 signals to array, then write out model and signal array to file, then clear signal array, increment signal index by 5, move index by 1, basecall index by 2.

Below are some more events, and below it how this moves the sliding window.

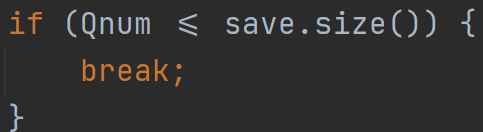




#### Printing to the output file

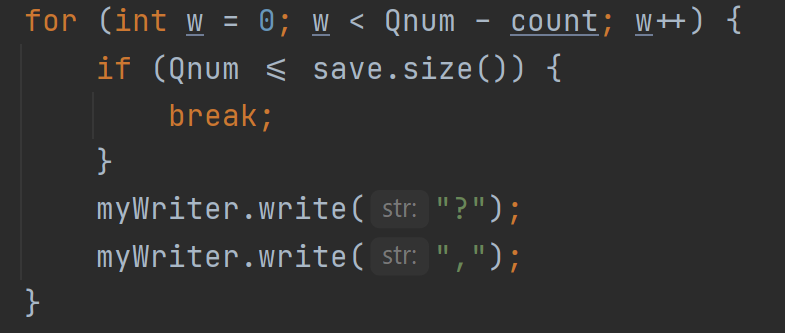
Each time a move is occurred, the 5 bases in the model and all the raw data in the save array list are printed as a data instance into the output file as the ARFF requirements. However, I mentioned earlier that not all bases are created under the same amount of raw data. And I also mentioned that WEKA requires all instances to have the same number of attributes. As attributes are the raw data values, this is problematic. To solve this, WEKA allows attributes to be marked as missing data. This is done by entering a question mark in place of the value. This means that at the beginning of the program it is possible to set the maximum value for the number of attributes each instance can have. Instances that are out of bounds for this value are not printed.

Save is the array list holding the current raw data values

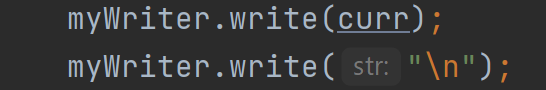


Values with equal or lower than the max value will be printed, and after printing an instance it will be counted how many attributes short was this instance, and the required number of question marks separated by commas are written at the end of the instance, before the 5 bases are printed at the end as the class.

Qnum is the pre-set number for the set number of attributes, and count is the current size of the array list where the current raw data are saved. This is printing the question marks.



And this is printing the class at the end of the instance



As soon as all of the files found in the folder are exhausted, the function returns the path of the program that it got from the bash script previously to the main function where it was called.

## WekaDemo class

The WekaDemo class is based on an example (10) on WEKA’s official website.

It begins with two functions where attributes classifier, filter to use is set. In the third function the file to read the data from is given. This file is the file that my code in CDImpute generates. Then in the fourth function the model to evaluate the classifier is built, in this instance it uses a 10-fold evaluation method. Lastly the toString method is overridden so that it gives back the results of the run.

## Chooser class

The Chooser class is based on JPanel, which is a graphical Interface class of Java. It is used to provide buttons where the user can choose from various options regarding the classification. At the moment it only allows users to choose between classification algorithms.

The class itself is where the buttons are added, and where it is defined what each buttons do.

The program is not complete, the original idea was to integrate support for the WEKA tool DeepLearning4J, which would allow building custom neural networks for the purpose of basecalling. However the time scope of this project proved to be much shorted then the time this could be achieved in. If I were to take this project further, I would be building a new class instead of the WekaDemo class, based on Deeplearning4J instead of standard WEKA class. There are a few examples of such classes designed for other uses (7).

## Running the program

To use my program, the prerequisites are that you are running Linux, and have h5dump (HDF Group) installed, h5dump is part of the hdf5tools Linux command line package.

You will also need to have at least Java version 58 installed.

In the package, you will have to make the start.sh file executable, using the “chmod +x start.sh” method. Then all the fast5 files that are to be used have to be placed in the folder called “files”. You will have to change directory into the program’s directory using the cd command, because this is what tells the program its current place in the system. Then you can execute the program by running “sudo bash start.sh”.

# Results and Conclusions

The initial goal of the project was to evaluate certain algorithms for use with basecalling.

Basecalling however turned out to be a much much more complex procedure that we initially thought it to be. The research papers that are available describe ideas and the methods used, as well as problems, but they do not go into detail on the actual structure of the neural networks employed. Sometime after being halfway through the project, it became clear that the original goal is out of scope due to the limited amount of time remaining and the complexity of the basecalling process. The biggest difficulty was to be able to bridge the gap between the file formats used by nanopore sequencers, and the format that WEKA uses. This difference is not only about a difference in the file format, as a difference also exists in the concept, or model used to represent the data. Therefore, most of my project’s work was about building a model of the data in fast5files that represents the information in a way that can be used in WEKA. I built about 5 different models when I was finally satisfied with the results.

After realising that the original goal is not obtainable, I decided to expend the code I already had into a program for Linux that can evaluate the performance of basecalling algorithms. However for reasons mentioned before, I was not able to build models of neural networks for basecalling. Instead, the program uses basic classification algorithms from WEKA. The program is functional using these. Such an example is the J48 decision tree, and the Naïve Bayes algorithm classifier. However these basic WEKA algorithms were not made for basecalling, so comparing these in terms of basecalling has no point really. This clearly shows when looking at the results, the accuracy tops out at about 1.7%. I assume that it tops out as when looking at the accuracy compared to the amount of training data used, no improvement can be seen, in some cases actually the accuracy gets lower with more training data. Based on this it is to be expected that increasing the amount of training data exponentially to an amount that can only be run on supercomputers will have little to no effect, and if it were to have an effect, using this would be super inefficient. This program is something that I could use to complete my original task, if I were to take this project further. How I would go about this is the Deeplearning4J toolkit to make a Java class with my own neural network suited for base calling, and then evaluate its performance and make changes, then evaluate it if it improved results, and then repeat these steps.

# Annotated Bibliography

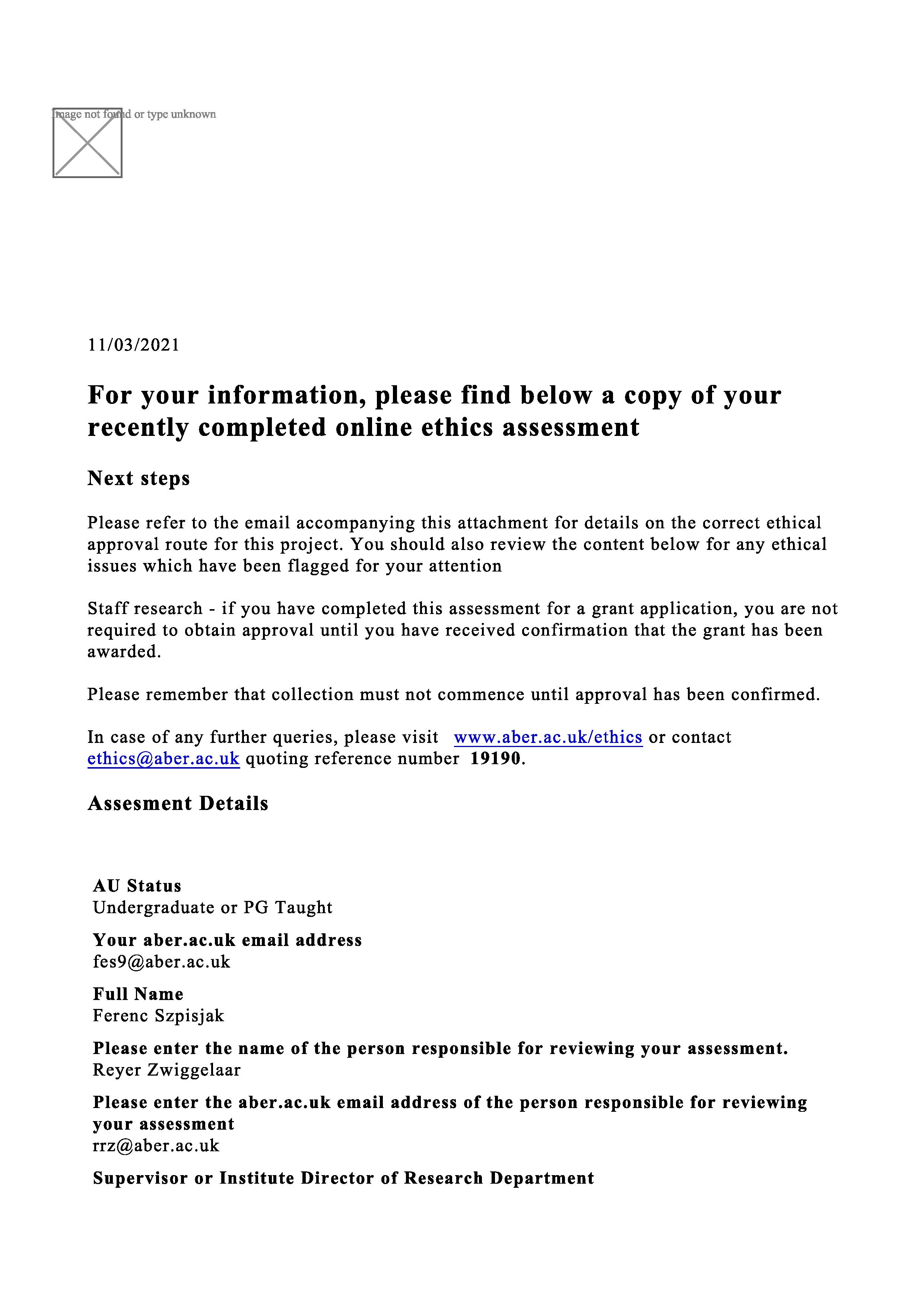
1. Haotian Teng, Minh Duc Cao, Michael B Hall, Tania Duarte, Sheng Wang, Lachlan J M Coin, Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning, *GigaScience*, Volume 7, Issue 5, May 2018, <https://doi.org/10.1093/gigascience/giy037>
2. Zhang, Yz., Akdemir, A., Tremmel, G. et al. Nanopore basecalling from a perspective of instance segmentation. BMC Bioinformatics 21, 136 (2020). <https://doi.org/10.1186/s12859-020-3459-0>
3. Boža V, Brejová B, Vinař T. DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads. PLoS One. 2017 Jun 5;12(6):e0178751. doi: 10.1371/journal.pone.0178751. PMID: 28582401; PMCID: PMC5459436.
4. Alexander Payne, Nadine Holmes, Vardhman Rakyan, Matthew Loose, BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files, *Bioinformatics*, Volume 35, Issue 13, 1 July 2019, Pages 2193–2198, <https://doi.org/10.1093/bioinformatics/bty841>
5. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.
6. Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0.
7. See appendices
8. Jain, M., Olsen, H.E., Paten, B. et al. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. Genome Biol 17, 239 (2016). <https://doi.org/10.1186/s13059-016-1103-0>
9. See appendices

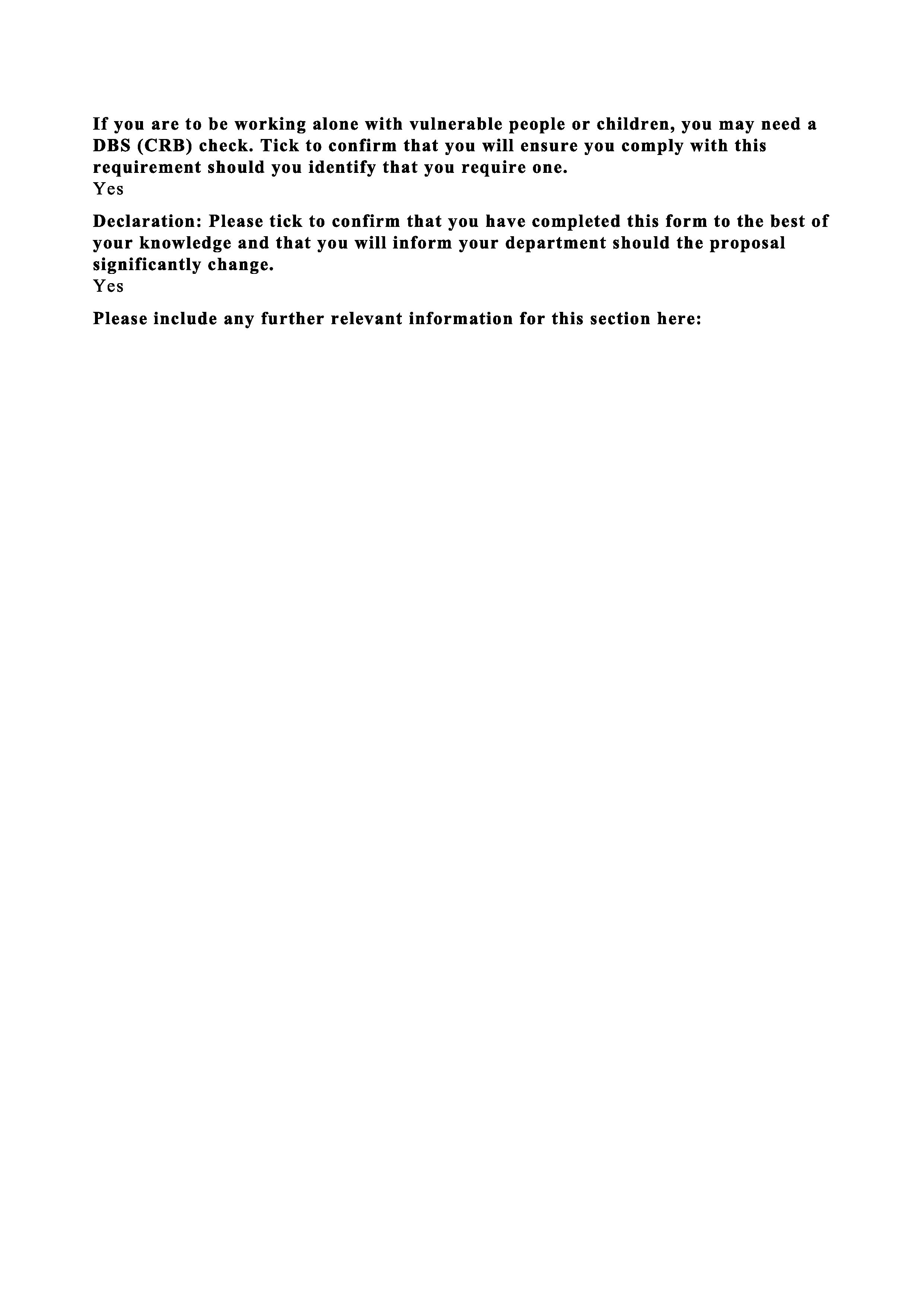
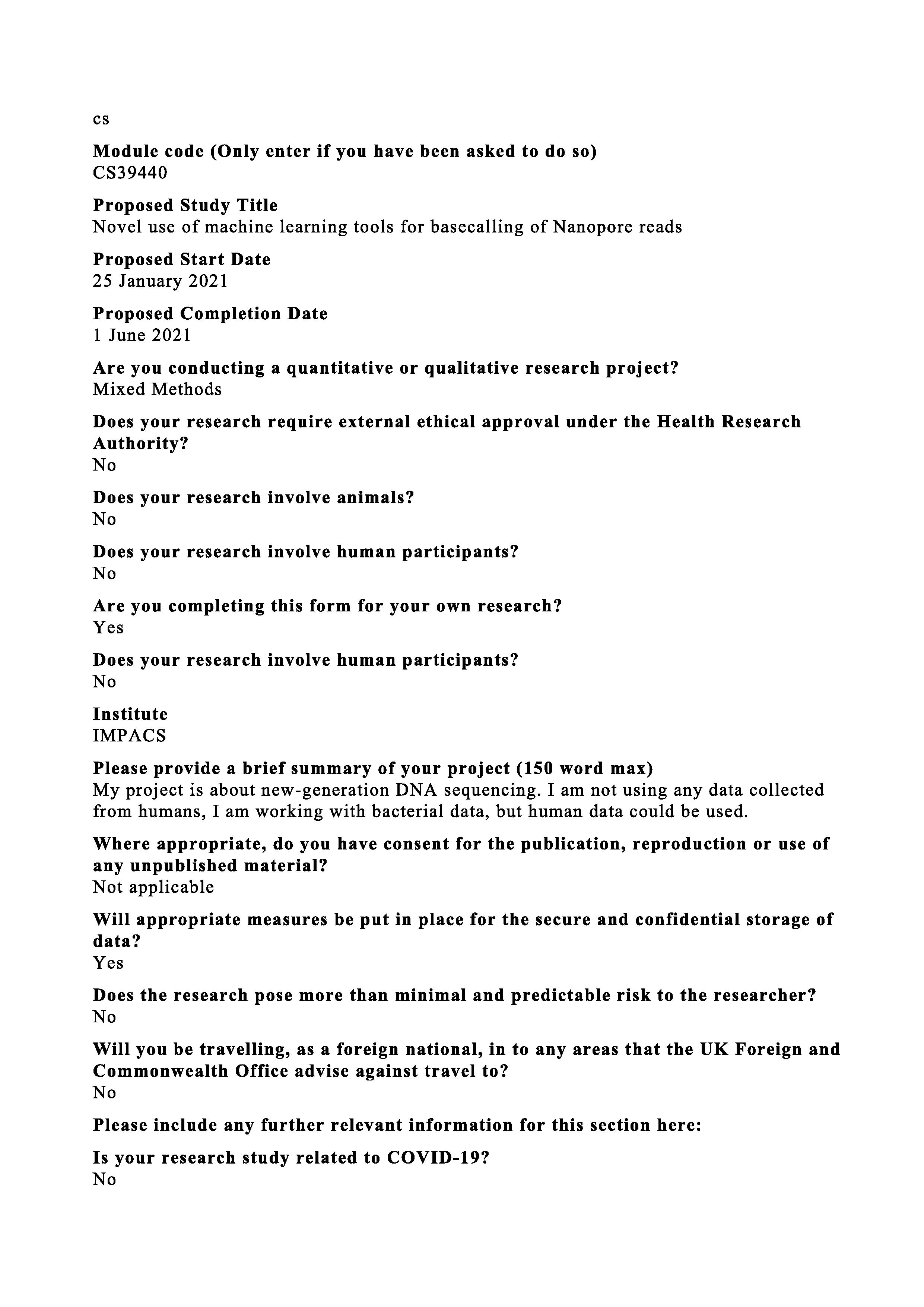
# Appendices

* 1. Third-Party Code and Libraries

WEKA Demo (10) class – It relies the WEKA jar file as a source file. The original class is used to run WEKA classification from a Java terminal, you have to pass the path to the input file as a system argument. I modified so that it receives the path of the file that is generated by my program. I also modified the algorithms it calls from WEKA, and the filter it uses for classification. To be able to select an algorithm I added code to use my Chooser class. This class available as an example of the usage of WEKA as part of Java code at <https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/>

Chooser (7) class – It is based on Java’s JPanel built in class. This code is originally used to select a file. I modified it so that instead of selecting a file the buttons modify a string, that is used to determine the classifier. Although it is modified quite a bit, the original example is from https://stackoverflow.com/questions/10083447/selecting-folder-destination-in-java/43138765

* 1. Ethics Submission



* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

Random Number Generator

The Bayes Durham Shuffle ensures that the pseudo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs.

// Some example code here…