UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 1415

Computing the Error Profiles of Third Generation Sequencing Technologies

Lucija Megla

Zagreb, July 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1415

Pristupnik: **Lucija Megla (0036471339)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Algoritam za računanje profila pogreške za tehnologije za sekvenciranje treće generacije**

Opis zadatka:

Razviti algoritam za računanje profila pogreške za očitanja koja generiraju uređaji za sekvenciranje tvrtki Pacific Biosciences i Oxford Nanopore Technologies. Očitanja i dulje sastavljanje sekvence poravnati na referentne genome koristeći alate GraphMap i Edlib. Pomoću alata SAMtools poravnanja pretvoriti u format SAM (Sequence Alignment/Map) i generirati oznaku MD. Na temelju MD oznake identificirati nukleotide koji su krivo očitani i nukleotide koji su obrisani. Koristeći prošireni niz znakova CIGAR (The Compact Idiosyncratic Gapped Alignment Report) niz znakova odrediti nukleotide koji su zamijenili krivo očitanje i nukleotide koji su umetnuti. Posebnu pozornost posvetiti homopolimernim regijama. Predložiti strategiju za ispravak pogreške u očitanjima. Programski kod treba biti dokumentiran i javno dostupan preko repozitorija GitHub.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:

Izv. prof. dr. sc. Mile Šikić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
MASTER THESIS COMMITTEE

Zagreb, 3 March 2017

MASTER THESIS ASSIGNMENT No. 1415

Student: **Lucija Megla (0036471339)**
Study: Computing
Profile: Computer Science

Title: **Computing the Error Profiles of Third Generation Sequencing Technologies**

Description:

Develop an algorithm for computing the error profiles of reads generated by Pacific Biosciences and Oxford Nanopore Technology sequencers. Align reads and reconstructed genome sequences on reference genomes using Graphmap and Edlib tools. Convert the alignment SAM (to Sequence Alignment/Map) format and generate the MD tag with SAMtools. Based on the MD tag in the SAM format identify the nucleotides that were replaced during the substitution and the types of nucleotides affected during the deletion. From the extended CIGAR (The Compact Idiosyncratic Gapped Alignment Report) string determine the substituting nucleotides and detect the nucleotides involved in an insertion. Pay special attention to homopolymer regions. Propose an error correction strategy. The complete application should be hosted on Github under an OSI-approved licence.

Issue date: 10 March 2017
Submission date: 29 June 2017

Mentor:
Associate Professor Mile Šikić, PhD

Committee Chair:
Full Professor Siniša Srbljić, PhD

Committee Secretary:
Assistant Professor Tomislav Hrkać, PhD

I would like to thank my family and friends for a huge amount of support during my whole education. Also, I would like to thank Robert Vaser for help during making of this thesis. Thanks to Li Chenhao for introducing me to the topic of this thesis. Last, but not least, big thanks to my mentor Mile Šikić for guidance and support.

CONTENTS

1. Introduction

Rapid development of computer hardware and computer software helped the field of Bioinformatics become one of the fastest growing scientific areas of our time. Bioinformatics is a hybrid field that brings biology and computer science together. It exploits different software tools and methods in order to address different biological questions and problems.

One of the best known challenges in Bioinformatics is genome sequencing. Genome sequencing is a process of deciding the precise order of individual nucleotid bases – A (as Adenyne), T (as Thymine), C (as Cytosine), G (as Guanine) and U (as Uracil) – within DNA or RNA molecules. In other words, sequencing is translating life's chemical alphabet into human alphabet (1).

History of sequencing dates back to 1970s when Fred Sanger developed the first method for sequencing called Sanger's method. This method, along with the other methods developed until early 2000s, is often called *First generation sequencing* method. Sanger's method produced long individual reads with 99.9% accuracy, but on the other hand, was very expensive and impratical. With Human Genome Project, that took place in early 2000s, sequencing began its rapid development which led to upgraded sequencing techniques called *Second generation sequencing*. These methods produced short reads (up to 300 - 400 bp[1]).

Today, we are slowely introduced to *Third generation sequencing*. *Third generation sequencing* includes *Pacific Biosciences* and *Oxford Nanopore Technologies*. Reads sequenced by this methods are very long, but also very erroneous, that is, they have lower accuracy than other methods (between 87% and 97%) (2; 3).

In this thesis we are interested in *Oxford Nanopore Technologies*, particulary because they have certain limitations. It is said that these technologies have problems with determining homopolymer[2] lengths, especially when longer homopolymers occur (4). Here, we found our motivation for this thesis. We developed an algoritham for finding error profiles of homopolymers in reads aligned to the sequence. Since these errors mostly happen with Oxford Nanopore Technologies (ONT) (5), the results will mainly focus on analyzing reads sequenced by ONT. Furthermore, error profiles will be used in machine learning algorithms

[1] Base pairs: A-T, C-G for DNA; A-U, C-G for RNA.

[2] Definition is given in Chapter 2.

in order to see if it is possible to correct the errors.

This thesis is organized as follows:

In Chapter 2, a quick overview of biological and technical terms will be given. We will also explain what error profiles are and how reads, reference sequence and consensus sequence are connected.

In Chapter 3 there will be an overview of the algorithm developed, along with its pseudocode, implementation and complexity.

Chapter 4 will give an overview and short introduction to machine learning alongside methods that were used to fix errors in homopolymers.

After presentation of the algorithm and machine learning methods that were used, in Chapter 5 final results will be presented.

After introducing the results, Chapter 6 will give a short discussion in order to argue the outcome of this thesis.

Chapter 7 will be a conclusion of this thesis, where the work and progress alongside with possible improvements will be disclosed.

2. Prerequisites

Some basic biological and technical terms will be introduced in this chapter for better understanding of chapters following this section. Firstly, there will be an overview of basic biological terms that will clarify any ambiguities that may be connected with biological background. Also, the connection between reference sequence, consensus sequence and reads will be explained, which is extremely important for understanding following chapters. Additionally, file formats for storing biological data will be introduced. Last, but not least, we will mention ***graphmap*** and ***dnadiff***, two tools that were needed in certain parts of the whole process.

2.1. Biological terms

- **read** - a raw sequence of nucleotides that comes off a sequencing machine
- **ONT** - Oxford Nanopore Technologies[1]
- **reference sequence** - nucleotide sequence assembled by scientists as a **representative** example of a species' set of genes (6).
- **consensus sequence** - calculated order of most frequent nucleotides found at each position in a sequence alignment. It represents the results of multiple sequence alignments in which related sequences are compared to each other and similar nucleotide patterns are calculated (7).
- **sequence alignment** - way of arranging the sequences of DNA, RNA or protein to identify regions of similarity between the sequences
- **homopolymer** - polymer composed of the same nucleotides
- **sequencing** - deciding the precise order of individual nucleotid bases (A,T,C,G,U) within DNA or RNA molecules.

Reference sequence is the norm. When reads are assembled into a longer sequence and finally into a consensus, the goal is to reproduce a consensus that is as similar to the reference

[1] See Chapter 1

as possible. Unfortunately, neither sequence process nor assembly process is perfect. That is why consensus has errors and differs from reference sequence. Particularly, for ONT reads, these errors happen at homopolymer regions (3). In other words, there must be a way of determining the magnitude of an error. That is why error profiles are found.

Error profiles are nothing more than the frequency of 3-dimension tuples: *(nucleotide, reference_length, read_length)*. The 3D tuple denotes that a certain read homopolymer, built of *nucleotide*, has aligned to a homopolymer in the reference, where length of a read homopolymer was *read_length* and that of reference homopolymer was *reference_length*. These error profiles can be useful in process called base calling[2]. Furthermore, by extending error profiles with coordinates of a homopolymer in the reference sequence, we were able to create feature vectors for training machine learning algorithms and alter the consensus in order to come even closer to the original reference sequence.

As mentioned before, Chapter 3 will explain how error profiles were found and Chapter 4 will explain how feature vectors were made.

2.2. File formats

2.2.1. FASTA format

In bioinformatics, FASTA format is one of the most used file formats for keeping nucleotide sequences. It is a text-based format where a sequence of nucleotides begins with a single-line descriptor[3] followed by lines of sequence data. An example of FASTA format can be seen in Figure 2.1.

```
>gi|544886608|ref|NZ_AURB01000207.1| Alicyclobacillus acidoterrestris
GTTTTGGGAACGGGTATTGACAGGGAGTTTTGGGTACATGGATTACGCAAGGGCAGAAACACCCAGTTCC
ACATGGACGATATTGCTCTGACCGTCATTTTCGGTTCCTTGCTGGGTCAGGAACGGATTTTCCACTTTGA
GGACATCGAACAAGATCCCCTGTTGAAGCTGAAGTTGGACGTGCCGAAACTGCCTGATACGACTCTGTTG
>gi|220683588|gb|FJ158840.1| Jeotgalicoccus huakuii
AGAGTTTGATCCTGGCTCAGGATGAACGCTGGCGGCGTGCCTAATACATGCAAGTCGAGCGCGAGCGTAA
GGAGCTTGCTCCTTACAATCGAGCGGCGGACGGGTGAGTAACACGTGGGCAACCTACCCTTTAGACTGGG
ATAACTACCGGAAACGGTAGCTAATACCGGATAAGTTGGATTACACAAGTAATCTTAATGAAAGGCGGAT
TTATCTGTCACTAAAGGATGGGCCTGCGGTGCATTAGCTAGTTGGTGAGGTAGTGGCTCACCAAGGCAAC
```

Figure 2.1: FASTA format example

[2]process of assigning nucleotides to chromatogram peaks. In other words, base calling is the process by which an order of nucleotides is inferred (8; 9)

[3]Single-line descriptor containing a name of the sequence and possibly some general information about the sequence.

2.2.2. FASTQ format

Besides FASTA format, FASTQ format is also a commonly-used text-based format for storing biological sequences. The difference is that FASTQ format, alongside nucleotide sequence, also keeps and its corresponding quality scores. Quality score is an integer mapping of a probability that the corresponding nucleotide is incorrect (10). Both the sequence letter and quality score are each encoded with a single ASCII character. An example of FASTQ format can be seen in the Figure 2.2.

```
@FCD044UACXX:4:1101:1778:2233#CAGATCAT/1
CGTCTGCACACTTCGTGAGCCATTTCCGGATTGTGGCACAGTCTGTGGGTGCGTGATACATCTGACGACACTTGAAACTTAAAATTGGAA
+
^^W`\ccccc]cd\bdZaYedehhhdedU_cRcechhhaccacc^X^cF[ZZ`c_\HMHM^ccb`XZ][^```]ZY]`]`TY`Z]]R]]
@FCD044UACXX:4:1101:2733:2230#CAGATCAT/1
AGATAGATGAGAAGAGACATTTAAAATGTGAAGGAAGTTTCTGGGCTCAAAAGAATAGTATCTATCTCAGTCTGTGTAGGAAAGGATGTG
+
bbaeeeeegggggiiiiiiiiihiiiihhfhiiiighiiiiiiiiiiiiiiiiiieghiihiiiiihiiigbdgiiggggeeeee
@FCD044UACXX:4:1101:2648:2238#CAGATCAT/1
GGTCACATCTTTGAAGGTAATTGGGCCAGGTGGGCCTGGAGTGTCTAGCACTTTCACGGTGAATGTGATTGACTTACTACCACTGTTGTT
+
bbaeeeeegggggiiiidhhiiiihiiiibfhihihiifhfghhiihiiiiiiiiiighiiihdhegggggeeceeeeeddddcccccc
@FCD044UACXX:4:1101:3716:2245#CAGATCAT/1
CTCCTGCCTCCGTGCTGATTGCAGGCCCTGTTTCCCCCAGGACTCCATGGCTTCCGAGTTGCTGACTGACCCTCCACCTCAGAGGTAGTT
+
bbb^ceeeggggghiiiiiiiiiihiiiighiiiiihiihhiihiiihiiiihifhdggggeeeeeeddddcccbccccbcc]`bbd
@FCD044UACXX:4:1101:4203:2232#CAGATCAT/1
CGGGTCTTTTCTCCCCCTCCCCATATGAAGCATCAGGGCTGAGCCCAGGGTCCGGGGGGAGGGGAGGGAGGCACAGCGGTGCTTCTTGGGG
+
__bea_ccggcgafhhhh`ghffd^ded_^cefef]dfhhffgffhhhd`_H\aYW^^WZVVaVWLTOTXWQW`baX]T[abbbb_RY^]
```

Figure 2.2: FASTQ format example

2.2.3. SAM format

SAM format is a commonly-used text-based format for storing aligned nucleotide sequences in a series of tab delimited ASCII columns. SAM stands for *Sequence Alignment/Map* format and it consists of arbitrary header section and an alignment section. Header lines start with @ (if they exist). Each alignment line has eleven mandatory fields which contain essential alignment information and volatile number of optional fields for aligner-specific information (11). Here we can examine mandatory fields of SAM file

1. **QNAME** - Name of the read, *String*
2. **FLAG** - Combination of bitwise FLAGs describing read or alignment specifics, *Int*
3. **RNAME** - Reference sequence name, *String*
4. **POS** - 1-based leftmost mapping position of the first matching base (beginning of the alignment), *Int*
5. **MAPQ** - Mapping quality, *Int*
6. **CIGAR** - The CIGAR string of the alignment, explained deeply later, *String*
7. **RNEXT** - The reference name of the next read, *String*

8. **PNEXT** - The position of the next read, *Int*
9. **TLEN** - Observed length of the template, *Int*
10. **SEQ** - The read sequence, *String*
11. **QUAL** - ASCII encoded base qualities, *String*

The important fields for this thesis are POS and CIGAR, as well as optional field that follows TAG:TYPE:VALUE format, MD tag. Description of POS field mentioned above is straightforward, but CIGAR string and MD tag need more explanation.

CIGAR string

CIGAR string is a sequence of letters and numbers that describe an alignment of a read to a reference. When a read is aligned to a reference we can have several cases and the most important ones are given in a Table 5.3. Regex for CIGAR string is: *|([0-9]+[MIDNSHPX=])+*, which means that before every letter is a number denoting how many consecutive times the operation appears in the alignment. In Figure 2.3, one can examine read alignment to a reference sequence with CIGAR string (11).

Table 2.1: Important CIGAR string letters/operations

Operation	Description
M	alignment match
I	insertion to the reference
D	deletion from the reference
S	soft-clipping
H	hard-clipping
N	spliced read

```
reference:   AA-TGCA-TTGAGACTATAT-CCGTGT
read:        C-TAACAT--GAGA--TATAC--GTGT
CIGAR:       MDI4MI2D4M2D4MI2D4M
```

Figure 2.3: Alignment example with CIGAR string

MD tag

As mentioned before, MD tag is an optional field with TAG:TYPE:VALUE format. It is a sequence of letters, numbers and special characters matching the following regex: *[0-9]+(([A-Z]|[̂A-Z]+)[0-9]+)**. The MD field wants to achive SNP/Indel calling[4] without looking at the reference. Unfortunately, in order to be able to reconstruct alignment without a reference, one must use CIGAR string alongside MD tag (12).

2.3. Tools

2.3.1. Graphmap

Graphmap is a software tool for read alignment (13) and is peculiar because it is targeted at aligning long, error-prone third-generation sequencing data. Even though it was designed to handle Oxford Nanopore reads, it is also able to handle a wide range of reads with different properties (like length and error profiles), which makes it suitable for PacBio read alignment.

Graphmap was used to align reads along reference sequence and consensus sequence. The output file in SAM format was then used as an input for the algorithm for finding error profiles, which will be explained in detail in the next chapter.

2.3.2. dnadiff

Dnadiff is a software tool for comparison of two genomes (14). It provides detailed information and quantification on diversity between two genomes.

Dnadiff is used after we trained machine learning algorithms to fix the error in homopolymers, so we could verify the difference between our reference genome and our new, altered consensus. Dnadiff is excelent for this task, because it provides us with precise report. Dnadiff outputs many files, but the file that is of interest is **.report* file, which provides scores on average identity between genomes, that is, between reference and altered consensus. Detailed method about how we altered consensus will be explained in Chapter 4. An example of **.report* file can be seen in Figure 2.4.

The important fields in **.report* file are: *TotalBases*, *AlignedBases* and *AvgIdentity*. Field *TotalBases* shows the total length of the reference (left column) and some sequence query (right column). Sequence query length should be as close as possible to the length of the reference sequence. Field *AlignedBases* shows how many bases aligned between the reference

[4] SNP/Indel calling is a type of next generation sequencing analysis.

	[REF]	[QRY]
[Sequences]		
TotalSeqs	1	1
AlignedSeqs	1(100.00%)	1(100.00%)
UnalignedSeqs	0(0.00%)	0(0.00%)
[Bases]		
TotalBases	4641652	4683356
AlignedBases	4090036(88.12%)	4051435(86.51%)
UnalignedBases	551616(11.88%)	631921(13.49%)
[Alignments]		
1-to-1	5	5
TotalLength	4060575	4051549
AvgLength	812115.00	810309.80
AvgIdentity	99.31	99.31
M-to-M	52	52
TotalLength	4099900	4090810
AvgLength	78844.23	78669.42
AvgIdentity	99.28	99.28

Figure 2.4: Example of *.report file

sequence and sequence query. Field *AvgIdentity* shows how similar are two sequences. Both *AlignedBases* and *AvgIdentity* should have high values.

3. Algorithm for computing error profiles of homopolymers

As mentioned before, one of the goals of this thesis was to develop an algorithm that will manage to find error profiles of homopolymers. The algorithm is not sequence-technology specific, but the results will be aimed to ONT reads. In Figure 3.1 we can see an example of error profile described in Chapter 2. Homopolymer in the reference sequence has $length = 6$, while the homopolymer in *read 1* has $length = 3$. The error profile for this case is: ('A', 6, 3). Homopolymer in *read 2* has $length = 5$, so the error profile for this case is: ('A', 6, 5).

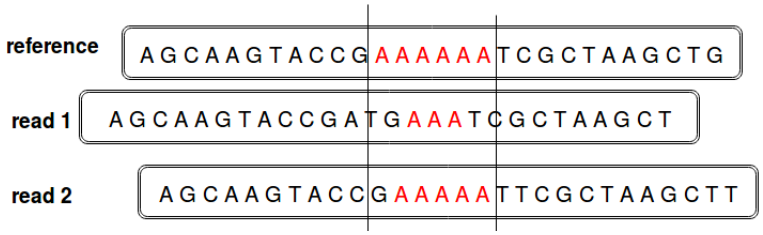


Figure 3.1: Homopolymer error example

Homopolymer regions are particulary interesting because third generation sequence technologies tend to have higher error rates when faced with longer homopolymers. This is especially true for Oxford Nanopore Technologies. That is why finding homopolymer regions in the reference and observing homopolymer regions in the reads aligned to the reference can give us extra information. This information can be used in sequencing methods.

In this chapter, an algorithm for finding error profiles will be discussed.

The algorithm is based on the idea of automaton. It iterates over the main sequence (reference or consensus sequence), reads aligned to the main sequence and CIGAR string simultaneously. CIGAR string is an important factor here, because it shows how pointers should be incremented. MD tag was an option to use instead of CIGAR string, because it contains information about nucleotides in the reference. That way there would be no need to load reference sequence in the memory. Unfortunately, in order to use MD tag, one should

also use CIGAR string. Parsing both and reconstructing the reference sequence would be too complex, so loading reference in the memory and using only CIGAR string seemed like a better solution.

During iteration, four states are interchanging. States track existance of homopolymers in the read and in the sequence. Their meaning is as follows:

- **State.on_on** - currently, there exists a homopolymer in the read and in the sequence
- **State.off_off** - currently, there doesn't exist a homopolymer in the read nor in the sequence
- **State.on_off** - currently, there is a homopolymer in the read, but not in the sequence
- **State.off_on** - currently, there is a homopolymer in the sequence, but not in the read

3.1. Pseudocode

Algorithm 1 Algorithm for finding homopolymer error profiles

```
1:  procedure HOMOPOLYMER_READ_ERRORS(reference, aligned_reads)
2:      reference ← genome_preprocessing(reference)
3:      error_dict ← ()
4:      regions_dict ← ()
5:      ref_max ← 0
6:      read_max ← 0
7:  loop:
8:      for read in reads do
9:          cigar ← read.cigar
10:         if read.cigar is None then
11:             continue;
12:         reference_pointer ← read.pos
13:         read_pointer ← 0
14:         cigar_pointer ← 0
15:         read_homopolymer_len ← 0
16:         reference_homopolymer_len ← 0
```

```
    ref_homopolymer_begin ← 0
    read_homopolymer_end ← 0
    state ← State.off_off
    while cigar_pointer < len(cigar) do
        if cigar[cigar_pointer] = 'M' then
            if state == State.off_off then
                if read_homopolymer_starts then
                    state ← State.on_off
                    if reference_homopolymer_starts then
                        state ← State.on_on
                else if reference_homopolymer_starts then
                    state ← State.off_on
            else if state == State.on_off then
                if read_homopolymer_continues then
                    state ← State.on_off
                    if ref_pointer == 0 and ref_homopolymer_starts then
                        state ← State.on_on
                else // save reference homopolymer and read homopolymer
                    state ← State.off_off
            else if state == State.off_on then
                if ref_homopolymer_continues then
                    if read_pointer == 0 and read_homopolymer_starts then
                        state ← State.on_on
                else // save reference homopolymer and read homopolymer
                    state ← State.off_off
            else if state == State.on_on then
                if read_homopolymer_continues and ref_homop_continues then
                    state ← State.on_on
                else if read_homopolymer_continues then
                    state ← State.on_off
                else if ref_homopolymer_continues then
                    state ← State.off_on
                else // save reference homopolymer and read homopolymer
                    state ← State.off_off
        else if cigar[cigar_pointer] = 'I' then
            if state == State.off_off then
```

```
53:                 if read_homopolymer_starts then
54:                     state ← State.on_off
55:             else if state == State.on_off then
56:                 if read_homopolymer_continues then
57:                     state ← State.on_off
58:                 else // save reference homopolymer and read homopolymer
59:                     state ← State.off_off
60:             else if state == State.off_on then
61:                 if read_homopolymer_starts then
62:                     state ← State.on_on
63:             else if state == State.on_on then
64:                 if read_homopolymer_ends then
65:                     state ← State.off_on
66:         else if cigar[cigar_pointer] = 'D' then
67:             if state == State.off_off then
68:                 if ref_homopolymer_starts then
69:                     state ← State.off_on
70:             else if state == State.on_off then
71:                 if ref_homopolymer_starts then
72:                     state ← State.on_on
73:             else if state == State.off_on then
74:                 if ref_homopolymer_continues then
75:                     state ← State.on_off
76:                 else // save reference homopolymer and read homopolymer
77:                     state ← State.off_off
78:             else if state == State.on_on then
79:                 if ref_homopolymer_ends then
80:                     state ← State.on_off
81:         else
82:             read_pointer++
83:             cigar_pointer++
```

Update of variables was left out of simplicity. The only important thing to note is:

- **cigar[cigar_pointer] ='M'** - both read and reference pointer are refreshed (along with cigar_pointer).
- **cigar[cigar_pointer] ='I'** - only read pointer is refreshed (along with cigar_pointer).
- **cigar[cigar_pointer] ='D'** - only reference pointer is refreshed (along with cigar_pointer).
- **all other combinations** - only read pointer is refreshed (along with cigar_pointer).

3.2. Implementation

SAM file with aligned reads was created using ***graphmap***. The algorithm was written in Python. The implementation depends on libraries numpy, pysam and biopython. The implementation of the algorithm is in *readerrors.py* file. *readerrors.py* consists of three important functions: *readerrors*, *make_test_csv* and *make_freq_csv*:

- *readerrors* - implementation of the algorithm. It accepts two files: sequence file and SAM file with aligned reads generated with ***graphmap***. It outputs maximum homopolymer length found, region_dict used in *make_test_csv* and freq_dict used in *make_freq_csv*.
- *make_test_csv* - function for constructing .csv file with feature vectors used later in machine learning algorithm.
- *make_freq_csv* - function for generating .csv file with error profiles.

The results of the algorithm will be discussed in Chapter 5.

3.3. Time complexity

The main factors in time complexity are CIGAR string and the amount of reads in the SAM file. When computing error profiles, algorithm iterates over each read in the SAM file and for each read iterates over a whole CIGAR string. If amount of reads is denoted with *n* and the average length of a CIGAR string is denoted by *m*, the time complexity of this algorithm is $O(nm)$. In Figure 3.2 we can see execution time of the algorithm depending on different amount of reads.

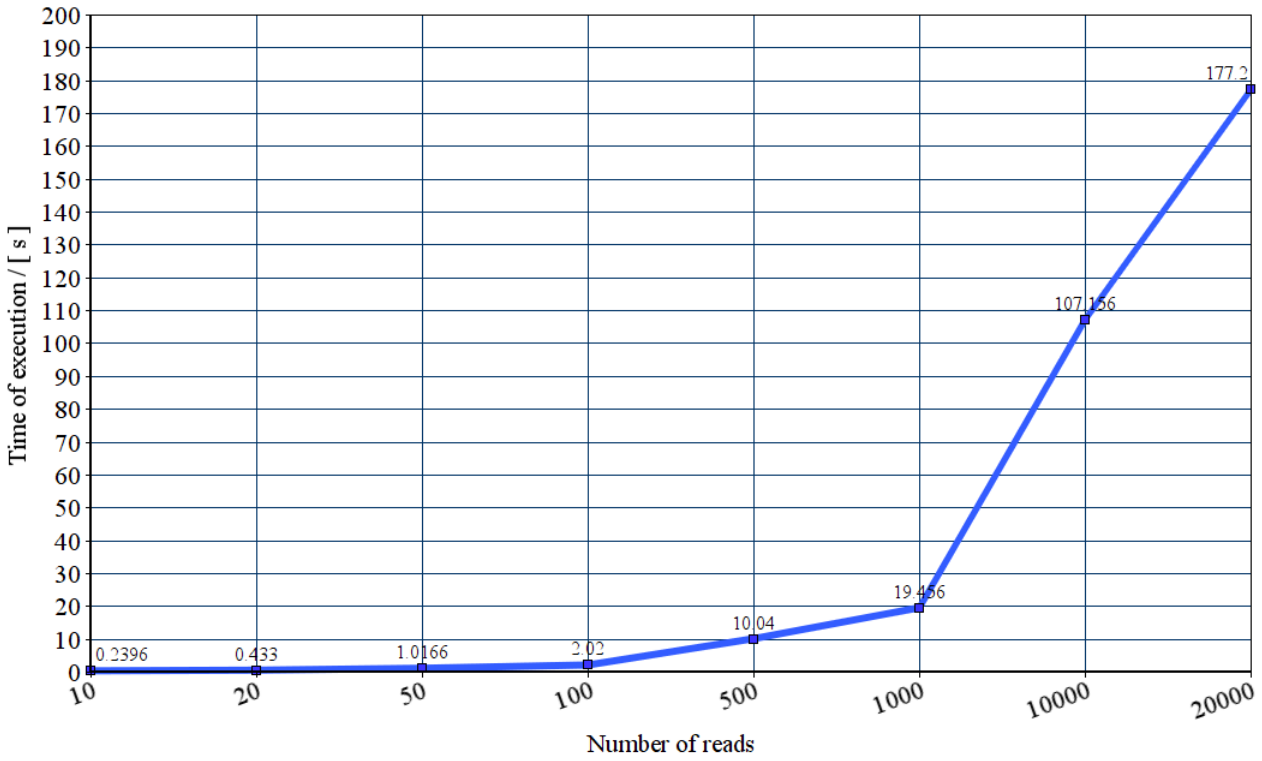


Figure 3.2: Time complexity for different number of reads

3.4. Reproducibility

In order to produce *train.csv* and *predict.csv* files needed for machine learning algorithms, first we had to create SAM files of reads aligned to the reference and reads aligned to the consensus sequence. After that, we executed *polisher.py* with SAM files, reference and consensus sequence as an input. Output of *polisher.py* are *train.csv* and *predict.csv* files. This

steps with exact commands can be seen in Figure 3.3

```
graphmap align -r <reference> -d <reads> -o <SAM_output_file_1>
graphmap align -r <consensus> -d <reads> -o <SAM_output_file_2>
python polisher.py -r1 <reference> -s1 <SAM_output_file_1> -c <consensus> -d <SAM_output_file_2>
```

Figure 3.3: Steps for getting files for machine learning algorithms

The code is freely available under MIT licence on `https://github.com/lucka318/master_thesis`.

4. Machine learning: error correction proposal

Machine learning is currently a hot topic in computer science. Because of technological improvements, machine learning today hugely differs from machine learning in the past.

Machine learning is a method of data analysis that automates analytical model building.[1] In other words, machine learning gives computers the ability to execute tasks without being explicitly programmed for them. Since machine learning has had huge success in today's world, a question arised: could machine learning improve quality of the consensus sequence?

As indicated before, the goal is to have the best approximation of reference sequence as possible. With help of algorithm for computing error profiles of homopolymers we built a dataset that will be used for training Support Vector Machines and Random Forests in order to predict the real length of homopolymer. Then, that homopolymer will be injected in consensus sequence for improvement.

In the next sections, feature vectors for this particular problem will be described, along with Support Vector Machines and Random Forests and their Python implementation.

4.1. Supervised learning

For this problem, supervised learning seemed as the best solution. Supervised learning is a type of machine learning where the task is to inferr a function from labeled training data in order to be able to map new examples and find out their values (15). In other words, there exists a training set that includes input data and their corresponding response values. Input data are often called feature vectors. There are two groups of supervised learning techniques: regression and classification. In our problem, we used classification, because homopolymer lengths can be divided between different classes.

Classification is a supervised learning category where data can be separated in *classes*. It is usually used for categorical response values.

Regression is a supervised learning category where output values are continuous and

[1] https://www.sas.com/en_us/insights/analytics/machine-learning.html

there is a certain relationship between the values.

4.2. Creating dataset

In order to get training examples that make sense and overall to correctly represent our data, be it for training, testing or mapping new examples, it is necessary to determine the shape and characteristics of feature vectors. In machine learning, a feature vector is an n-dimensional vector of numerical features that represent an object (16).

4.2.1. Feature vectors

In order to create the dataset, it was necessary to determine the look of feature vectors.

Each feature vector denotes a location of homopolymer in the reference (in case of training set). Length of feature vector (i.e number of features) is $n + 1$, where n is a maximum lenght of homopolymer that could be found in the reference sequence. Additionaly, each feature indicates a length of a homopolymer - from zero to maximum length of homopolymer in the reference. The value of each feature is a counter which marks how many homopolymers in the reads, aligned to the particular homopolymer in the reference, had the length denoted by the feature.

In Figure 4.1 we can see an example of this.

4.2.2. Data processing

After creating set of feature vectors labeled with the right class, we removed feature vectors where the sum of classes was less than 3. This was particularly because of cross-validation and split between training and test data. Since there was a huge amount of training data, classes which had 3 (or less) feature vectors were considered outliers.

4.3. Algorithms

In this section, Support Vector Machines and Random Forests will be explained.

4.3.1. Support Vector Machines

Support Vector Machine or SVM is a supervised machine learning algorithm. SVM is usually used for classification problems, but it can also be used for regression problems (17). Also, SVM is an optimization problem. When given labeled training data (which makes it a

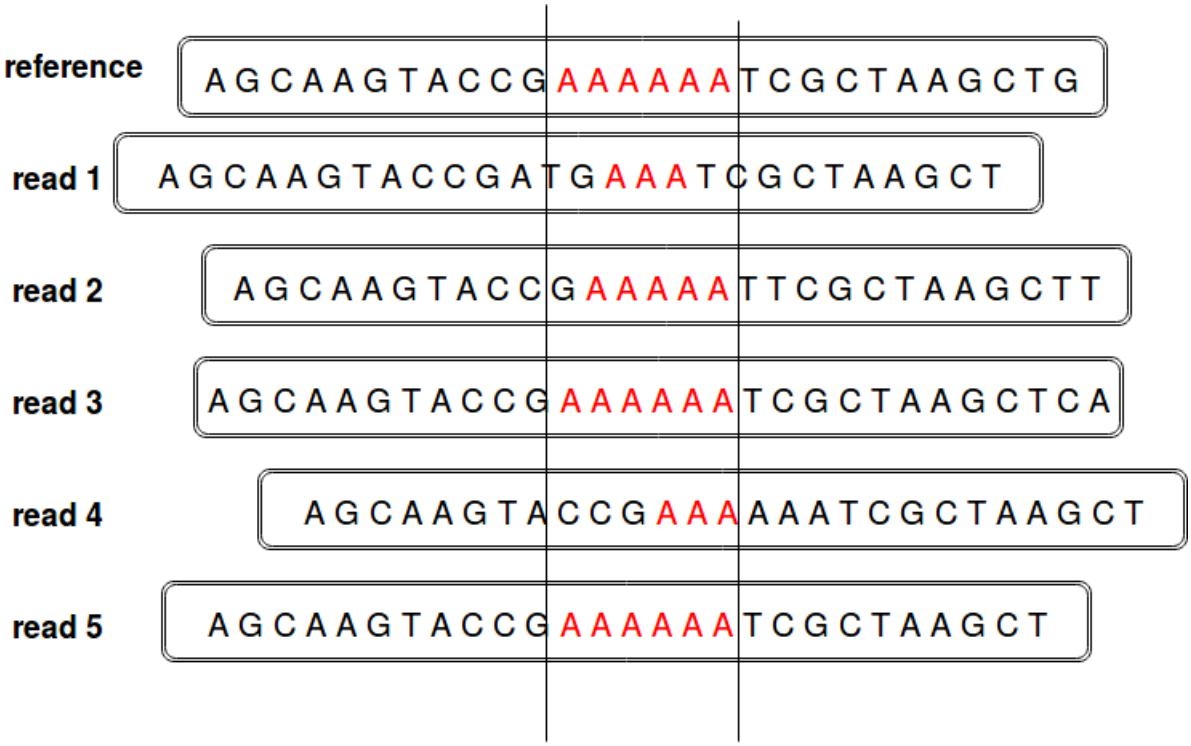


Figure 4.1: Feature vector example: In the figure there is a reference sequence with homopolymer AAAAAA and five reads aligned to it with their homopolymer lengths respectively: 3, 5, 6, 3, 6. Let us say maximum length of homopolymer in the reference is 7, then out feature vector will have 8 features. In this case the feature vector would be: $x = 0\ 0\ 0\ 2\ 0\ 1\ 2\ 0$ and $y = 6$.

supervised machine learning algorithm), the algorithm outputs an optimal hyperplane which classifies new examples (18). In figure 4.2 there is an example of Support Vector Machine.

In figure 4.2 there is an optimal hyperplane and three support vectors. The goal is to find a hyperplane that gives the largest minimum distance to the training examples. The largest minimum distance between the training examples of opposite classes is called a margin. Thus, optimal hyperplane maximizes the margin of the training data (18).

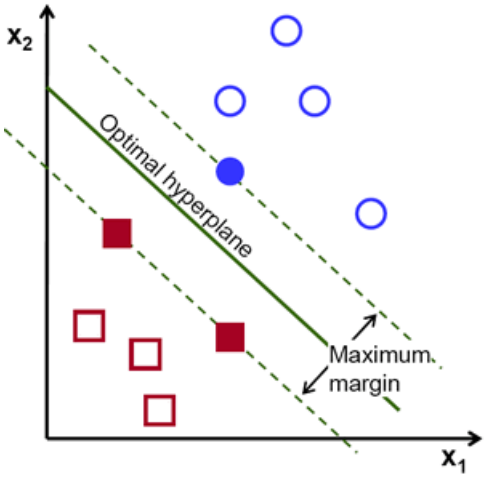


Figure 4.2: Support Vector Machine

Source: `http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`

4.3.2. Random Forests

Random forest can be used for classification and regression. It is an ensemble learning method. The main idea behind ensemble methods is to combine weak learners in a group to form a strong learner (19).

A weak learner in random forest is a decision tree machine learning algorithm. An example of decision tree can be seen in Figure 4.3 (20).

In the decision tree, internal nodes are tests on the attributes. A branch is an outcome of the test, while a leaf node denotes a class label (20).

The random forest algorithm combines decision trees to form a strong learner.

4.4. Implementation

Scikit-learn is a free software Python library for Machine learning (21). From scikit-learn we used *RandomForestClassifier* and *SupportVectorClassifier* for our classification problem. Also, we used *train_test_split* from *sklearn.model_selection* to split our dataset into a training set and test set. For choosing the best estimator, we used *GridSearchCV*.

4.4.1. train_test_split

Train_test_split is a method from *sklearn.model_selection*. It splits dataset into a random train and test sets. The main parameters that were used in the implementation were (21):

- *estimator* - machine learning algorithm, we used SVC and RFC.

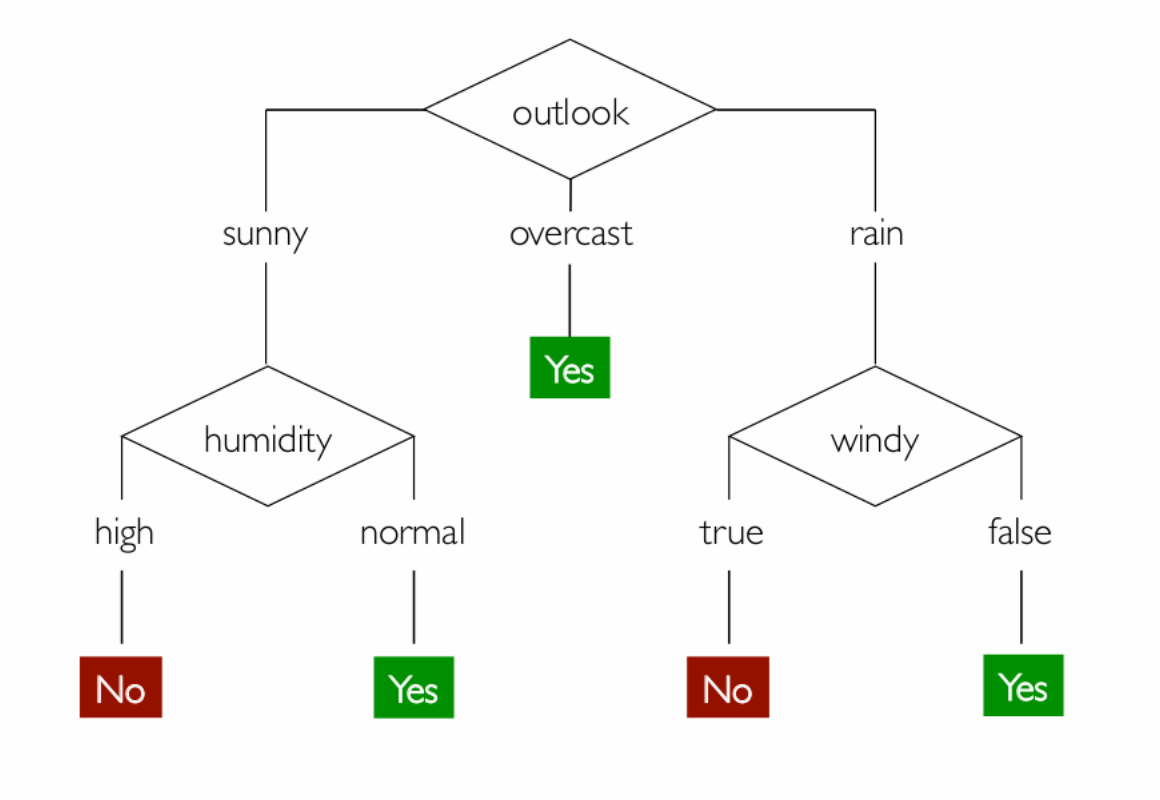


Figure 4.3: Decision tree

- *param_grid* - map of parameters for an estimator.
- *stratify* - this attribute makes a split in a way that the proportion of values in the sample produced is the same as the proportion of values provided to attribute.

4.4.2. GridSearchCV

GridSearch is a class which is used for exhaustive search over specified parameter values for an estimator (22). Basically, it explores the space of given parameters for a machine learning algorithm, chooses the best parameters for the estimator and fits the data using those parameters. he main parameters that were used in the implementation were (22):

- *test_size* - represents proportion of the dataset to include in the test split. Our test set was 30% of the whole dataset (*test_set=0.3*).
- *random_state* - pseudo-random number generator state used for random sampling.
- *n_jobs* - number of jobs to run in parallel. We used 12.
- *error_score* - value to add to the score if an error happens in estimator training. We used 0.
- *iid* - shows if data is identically distributed across the folds. In our case that is not true.

4.5. Output and altering consensus

After training the estimators, we aligned the reads to the consensus sequence, so we would get feature vectors for homopolymers in the consensus. The dataset was then given to the trained estimators to predict the homopolymer length in the consensus sequence.

When we had new values for homopolymer lengths, we implemented function for altering the original consensus. Basically, new consensus was created and in indices where homopolymers were, we added homopolymers with new lengths. The implementation is in *alter_consensus.py*

4.6. Reproducibility

In order to reach the altered consensus, first step is to train machine learning algorithms on train.csv file. This file was one of the output files from *polisher.py*. This file was made out of reads aligned to the reference sequence. After training phase, we predicted response values in *predict.csv* file. Data in *predict.csv* file came from reads aligned to the consensus sequence. The output of this phase are two .csv files: *predict_SVM.csv* and *predict_RFC.csv*. These two files contain predicted homopolymer lengths for consensus sequence. The last step is to call *alter_consensus.py* to alter the consensus. Then, we get two new, altered consensus sequences that have to be compared to the reference sequence in order to see how they dffer from the old consensus and to see qhich classifier preformed better. This steps with exact commands can be seen in Figure 4.4.

```
python train_classifiers.py -t train.csv -n <int> -v <int>
python predict_classifier.py -p predict.csv
python alter_consensus.py -c <consensus> -s <predict_SVM.csv | predict_RFC.csv> -o <output_altered_consensus>
dnadiff align <reference> <output_altered_consensus>
```

Figure 4.4: Steps to get altered consensus

The code is freely available under MIT licence on `https://github.com/lucka318/master_thesis`.

5. Results

In this chapter we will present our results, give a comparison between classifiers and see which one is better for our problem. In the next chapter we will discuss the given results.

5.1. Algorithm for finding error profiles

One of the tasks of this thesis was to compute the error profiles of homopolymers. We computed error profiles for *escherichia coli r7* reads, *escherichia coli r9* reads and *scerevisiae* reads. All reads were aligned to the reference and to the consensus and were sequenced by Oxford Nanopore Technologies.

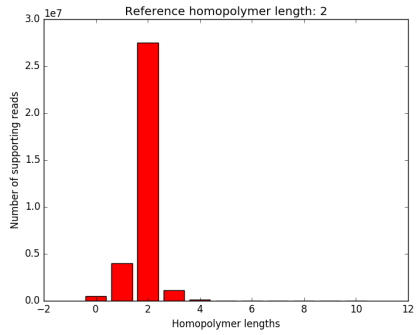
5.1.1. Escherichia coli r7 and Escherichia coli r9 reads

E. coli r7 and E. coli r9 reads are reads that are sequenced with different nanopore chemistries. The error profiles for E. coli r7 reads aligned to E. coli reference sequence can be seen in Figure 5.1. Every figure shows one class of homopolymer length found in the reference and the frequency of homopolymer lengths in the reads that were aligned to the homopolymer in the reference. What is interesting in these error profiles is that we can see how base caller cannot predict homopolymers longer than 6 when sequencing reads. If we take a look at Figure 5.1g or Figure 5.1h, there were no homopolymers in the reads that were longer than 6, even though length of homopolymer in the reference was 8 and 9 respectively.

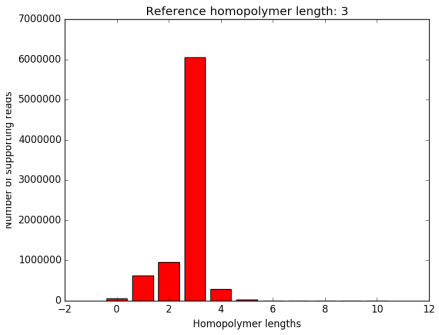
In Figure 5.2 we can see error profiles for E. coli r9 reads aligned to E. Coli reference sequence. What we can notice is that r9 nanopore technology in case of E. coli cannot base call homopolymers longer than 5, which is really interesting to see.

5.1.2. Scerevisiae r9 reads

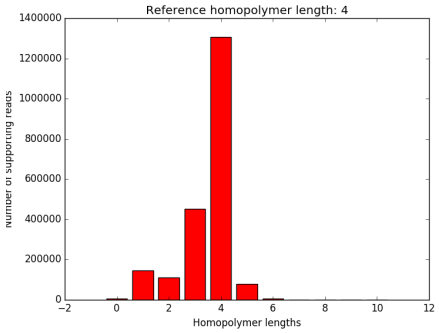
The error profiles for Scerevisiae ONT r9 reads aligned to Scerevisiae reference sequence can be seen in Figure 5.3. Since there are more different homopolymer lengths in scerevisiae than in E.coli, some figures were ommited. In case of Scerevisiae, we can observe that base caller is not consistent when determining homopolymer lengths. It base calles many dif-



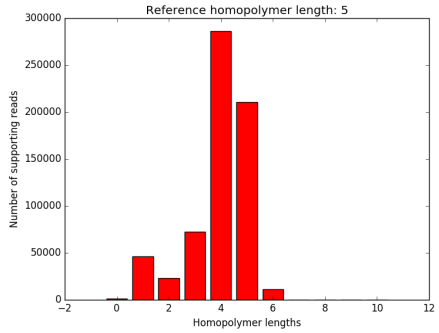
(a) Length of reference homopolymer equals 2



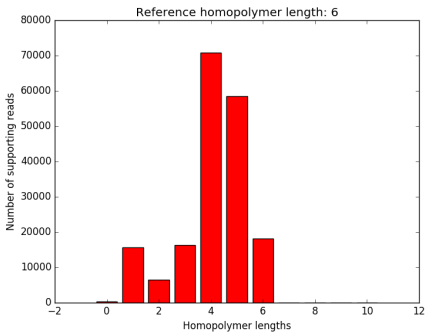
(b) Length of reference homopolymer equals 3



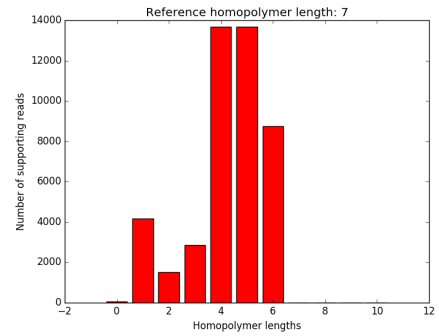
(c) Length of reference homopolymer equals 4



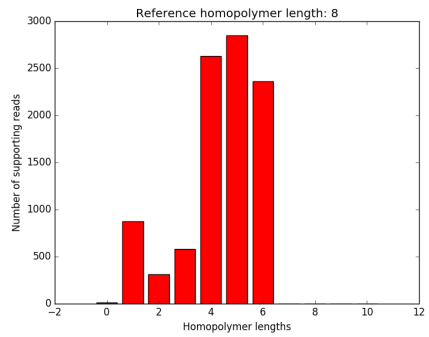
(d) Length of reference homopolymer equals 5



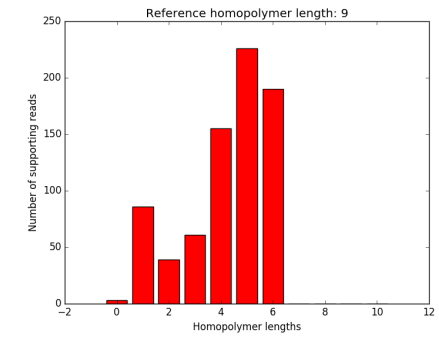
(e) Length of reference homopolymer equals 6



(f) Length of reference homopolymer equals 7

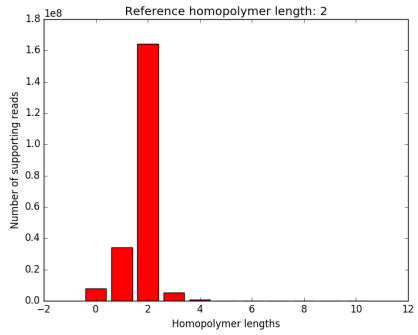


(g) Length of reference homopolymer equals 8

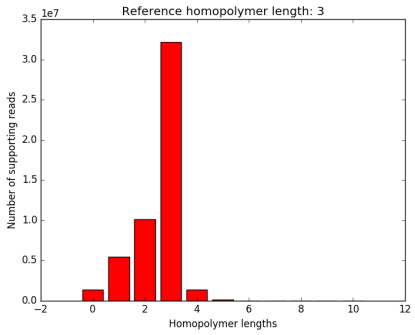


(h) Length of reference homopolymer equals 9

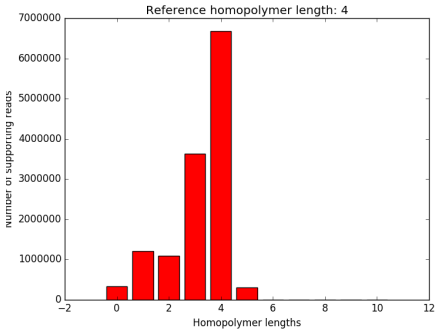
Figure 5.1: Error profiles for E. coli r7 reads aligned to E. coli reference



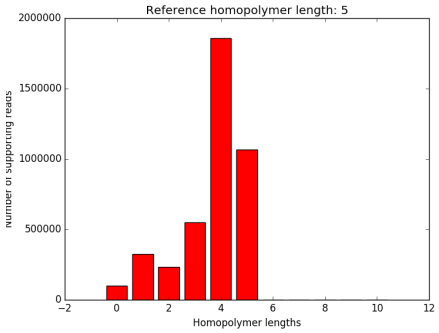
(a) Length of reference homopolymer equals 2



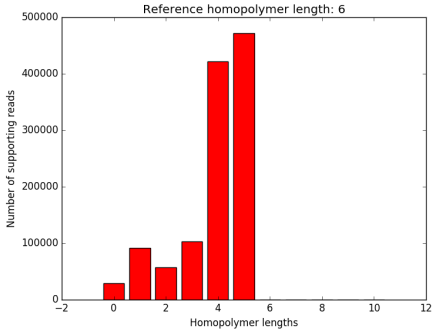
(b) Length of reference homopolymer equals 3



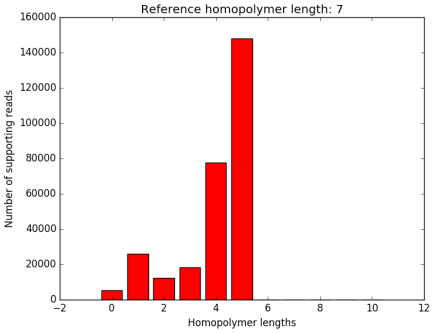
(c) Length of reference homopolymer equals 4



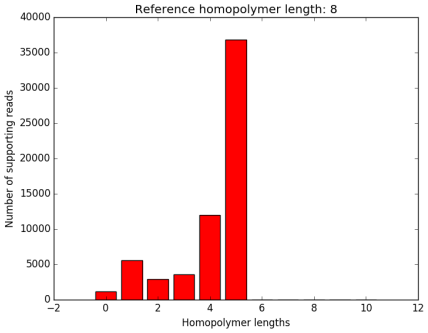
(d) Length of reference homopolymer equals 5



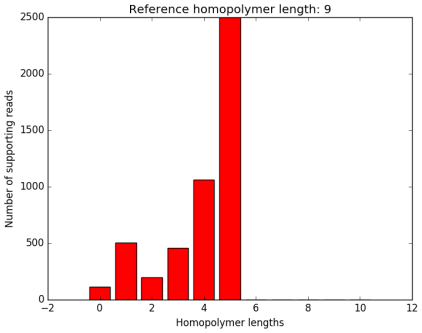
(e) Length of reference homopolymer equals 6



(f) Length of reference homopolymer equals 7



(g) Length of reference homopolymer equals 8



(h) Length of reference homopolymer equals 9

Figure 5.2: Error profiles for E. coli r9 reads aligned to E. coli reference

ferent homopolymer lengths, which can be problematic during training of machine learning algorithms.

5.2. Error correction

Table 5.1: Error correction results for E. Coli.r7

Dnadiff attributes	Reference	Before[1]	SVC[2]	RFC[3]
TotalBases	4641652	4632058	4640821	4647273
Aligned bases	-	4632055	4640810	4647267
AvgIdentity	-	99.32	99.49	99.25

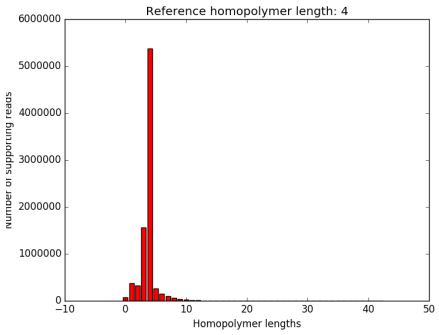
Table 5.2: Error correction results for E. Coli.r9

Dnadiff attributes	Reference	Before[4]	SVC[5]	RFC[6]
TotalBases	4641652	4604706	4627472	4619919
Aligned bases	-	4604705	4627470	4619919
AvgIdentity	-	98.76	98.84	98.99

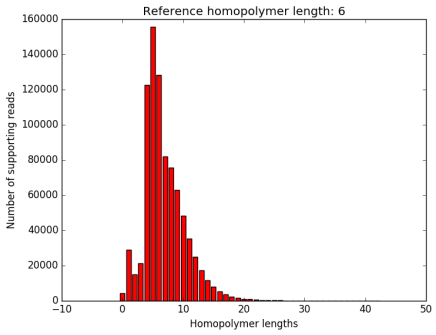
Table 5.3: Error correction results for Scerevisiae

Dnadiff attributes	Reference	Before[7]	SVC[8]	RFC[9]
TotalBases	12157105	12167721	12701510	12688138
Aligned bases	-	1622288	96885	109079
AvgIdentity	-	96.75	94.65	94.52

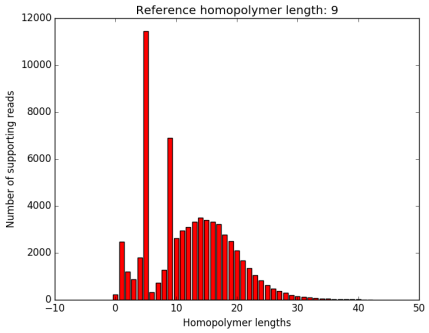
[3]Results of the original consensus
[3]Support Vector Classifier
[3]Random Forest Classifier



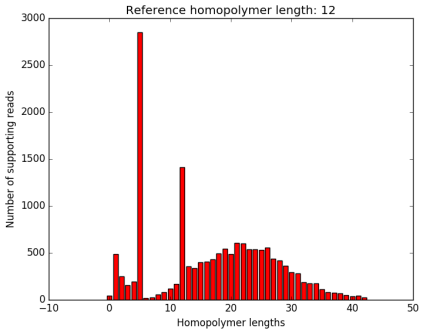
(a) Length of reference homopolymer equals 4



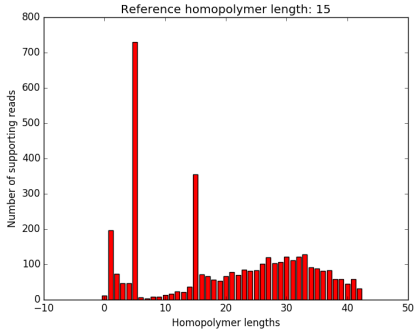
(b) Length of reference homopolymer equals 6



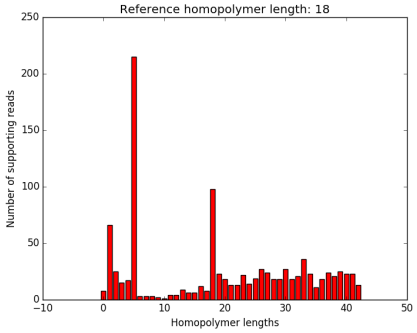
(c) Length of reference homopolymer equals 9



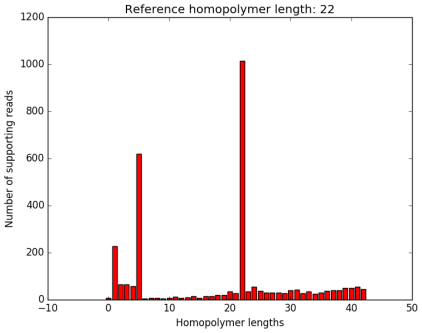
(d) Length of reference homopolymer equals 12



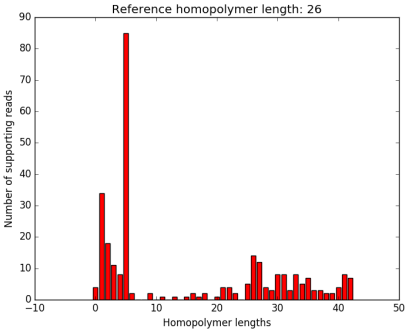
(e) Length of reference homopolymer equals 15



(f) Length of reference homopolymer equals 18



(g) Length of reference homopolymer equals 22



(h) Length of reference homopolymer equals 26

Figure 5.3: Error profiles Scerevisiae r9 reads aligned to Scerevisiae reference

6. Discussion

In previous chapter we saw the error profiles of homopoylmers and how consensus can be corrected using machine learning. In Figures 5.1, 5.2 and 5.3 it can clearly be seen how base caller of Oxford Nanopore Technologies has problems with determining the length of homopolymers that are greater than 6. Thus, when consensus is assembled, there is a low chance that the length of a longer homopolymer will be correctly assembled and resolved.

To fix this problem we tried using machine learning as the solution. In case of E.coli, machine learning managed to fix the consensus for the better. E.coli as a species did not have homopolymers longer that 10 which probably was a mitigating circumstance, because it is easier to generalize over a smaller group of possibilities. In case of E. coli r7 SVC managed to increase *AvgIdentity* for 0.17% and increase size of *AlignedBases* for 8763, that is, consensus came closer to refrence's number of *AlignedBases*. On the other hand, RFC did not show good results.

In the case of E. coli r9, RFC had better results than SVC, but both managed to improve older consensus. SVC fixed the *AvgIdentity* for 0.08%, while RBC fixed *AvgIdentity* for even 0.23%. Both RFC and SVC increased number of *AlignedBases* and came closer to reference's number of *AlignedBases*. On the other *scerevisiae* did not have good results. The case with *scerevisiae* is that it has homopolymer lengths up to 45. Since base caller cannot succesfully predict lengths greater than 6, this leaves a huge space for erroneous feature vectors. For example, as we can see in Figure 5.3h, even though the length of a homopolymer in the reference was 24, a great majority of reads had mixed homopolymer lengths, without any statistical trends in it. This was probably one of the reasons why machine learning could not fix the errors that happened.

When we compare SVC and RFC estimators, both gave good results in case of E. coli. Since SVC managed to improve both E. coli.r7 consensus and E. coli.r9 consensus, it could be a better estimator than RFC.

7. Conclusion

In this thesis we were interested in *Oxford Nanopore Technologies* for sequencing genome, particulary because they have certain limitations. It is said that these technologies have problems with determining homopolymer lengths, especially when longer homopolymers occur (4). That is why one of the goals was to develop an algorithm for finding error profiles of homopolymers.

The algorithm is based on the idea of automaton. It iterates over the main sequence (reference or consensus sequence), reads aligned to the main sequence and CIGAR string simultaneously. The main factors in time complexity of this algorithm are CIGAR string and the amount of reads in the SAM file. The time complexity of this algorithm is $O(nm)$, where the amount of reads is denoted by *n* and the average length of a CIGAR string is denoted by *m*. In Figure 3.2 we can see execution time of the algorithm depending on different amount of reads.

The next task was to propose a way to improve the consensus assembled from ONT reads. Since machine learning is very popular, we decided to try with classification algorithms. We created feature vectors that represented a location of homopolymer in the reference (in case of a training set). Length of a feature vector (i.e number of features) was *n + 1*, where *n* is a maximum lenght of homopolymer that could be found in the reference sequence. The value of each feature is a counter which marks how many homopolymers in the reads, aligned to the particular homopolymer in the reference, had the length denoted by the feature.

In Figure 4.1 we saw an example of this.

We decided to use *SupportVectorClassifier* and *RandomForestClassifier* from scikit Python library as our implementation for machine learning. *SupportVectorClassifier*, on average, had better results than *RandomForestClassifier*.

In case of *Escherichia coli*, we managed to improve the consensus, but in case of *Scerevisiae* did not. The problem with *Scerevisiae* reads is that read homopolymers had very irregular lengths, which probably is not favorable regarding machine learning.

Escherichia coli results can be a starting point in applying machine learning to sequencing problems. Next step could be to try and apply regression to this problem. Also, since *Scerevisiae* had problems with read homopolymers, it would be compulsory to further ex-

plore the dataset and feature vectors and try to model homopolymer features in a way that would be more robust and less sensitive to base caller's possibilities.

BIBLIOGRAPHY

[1] Genome News Network. *Genome sequencing*. J. Craig Venter Institut, 2003.

[2] Šikić M. and Domazet-Lošo M. Bioinformatika, 2013.

[3] Wikipedia. Dna sequencing. https://en.wikipedia.org/wiki/DNA_sequencing. [Online; accessed 12/06/2017].

[4] Ip L.C, Loose M., and J.R. Tyson. Minion analysis and reference consortium: Phase 1 data release and analysis. 2015.

[5] Robison K. Homopolymers and other recurring topics. http://omicsomics.blogspot.hr/2016/11/, 2016. [Online; accessed 12/06/2017].

[6] Wikipedia. Reference genome. https://en.wikipedia.org/wiki/Reference_genome. [Online; accessed 15/06/2017].

[7] Wikipedia. Consensus sequence. https://en.wikipedia.org/wiki/Consensus_sequence. [Online; accessed 20/06/2017].

[8] Genohub. Base calling. https://en.wikipedia.org/wiki/Consensus_sequence. [Online; accessed 20/06/2017].

[9] Dnabaser. Base calling for dna sequence/chromatogram files. http://www.dnabaser.com/help/snp%20mutation%20detection/base%20caller.html. [Online; accessed 20/06/2017].

[10] Wikipedia. Fastq format. https://en.wikipedia.org/wiki/FASTQ_format#Quality. [Online; accessed 26/06/2017].

[11] The SAM/BAM Format Specification Working Group. Sequence alignment/map format specification. 2017.

[12] Biostars. Is it possible to reconstruct alignment from cigar and md strings alone? https://www.biostars.org/p/112382/.

[13] Sikic M. and Sovic I. Graphmap. https://github.com/isovic/graphmap.

[14] S. Kurtz, A. Phillippy, A.L. Delcher, M. Smoot, M. Shumway, C. Antonescu, , and S.L. Salzberg. Dnadiff. https://github.com/garviz/MUMmer/dnadiff.

[15] Wikipedia. Supervised learning. https://en.wikipedia.org/wiki/Supervised_learning. [Online; accessed 21/06/2017].

[16] Pang H., Moore K., and Padmanabha A. Feature vector. https://brilliant.org/wiki/feature-vector/. [Online; accessed 21/06/2017].

[17] Ray S. Understanding support vector machine algorithm from examples (along with code). https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/. [Online; accessed 22/06/2017].

[18] OpenCV. Introduction to support vector machines. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html. [Online; accessed 22/06/2017].

[19] Blackwell A. A gentle introduction to random forests, ensembles, and performance metrics in a commercial system. http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics. [Online; accessed 22/06/2017].

[20] Lanzi P.L. Data mining and text mining: Classification: Decision trees. http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics. [Online; accessed 22/06/2017].

[21] Cournapeau D. scikit-learn. http://scikit-learn.org/stable/. [Online; accessed 25/06/2017].

[22] Cournapeau D. Gridsearchcv. http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Online; accessed 25/06/2017].

Computing the Error Profiles of Third Generation Sequencing Technologies

Abstract

When reads are assembled into a longer sequence and finally into a consensus, the goal is to reproduce a consensus that is as similar to the reference as possible. Oxford Nanopore Technologies have issues when sequencing homopolymers. That is why we developed a tool where we found error profiles of homopolymers. Using error profiles we trained machine learning algorithms to predict the real length of homopolymers. Then we altered the consensus accordingly, in order to reach reference sequence.

Keywords: homopolymers, error profiles, Oxford Nanopore Technologies, machine learning

Algoritam za računanje profila pogreške za tehnologije za sekvenciranje treće generacije

Sažetak

Kada se očitanja sastavljaju u duži redoslijed nukleotida i napokon u konsenzus, cilj je reproducirati konsenzus koji je što sličniji referenci. Oxford Nanopore Technologies imaju problema kod sekvenciranja homopolimera. Stoga je razvijen alat koji računa profile pogreške homopolimera. Koristeći profile pogrešaka, istrenirani su algoritmi strojnog učenja kako bi predvidjeli stvarnu duljinu homopolimera. U skladu s tim, konsenzus je mijenjan kako bi bio što sličniji referenci.

Ključne riječi: homopolimeri, profili pogrešaka, Oxford Nanopore Technologies, strojno učenje