Comparison of VCF files

May 24, 2021

1 Introduction

The goal of this assignment is to do variant calling on human_g1k_v37_decoy.fasta reference genome and C835.HCC1143.2.converted.realigned.base_recalibrated.bam file using two callers: GATK 4 HaplotypeCaller [1] and FreeBayes [2].

After that, results need to be compared using HaplotypeCaller's one as a truth set and Freebayes' one as a test set. Information of interest is: number of true positives, false positives and false negatives, precision, recall and f-score.

2 Variant calling

In order to do variant calling with GATK 4 HaplotypeCaller, two input arguments are required: input alignments (BAM file) and reference (FASTA file). Apart from those, two additional files need to be present in the directory with reference files, with FAI and DICT formats. Results are written into gatk_result.vcf, and this will be considered the truth set of further analysis.

FreeBayes also requires same arguments as previous caller, but it does not require any additional files to be present. Results are written into freebayes_result.freebayes.vcf, which will be test set of further analysis.

3 VCF comparison

VCF is a text file format that contains meta-information lines, header line, and then data lines each containing information about a position in the genome. Format also has the ability to contain genotype information on samples for each position. [3] It is the format in which the results of both previously used tools were written.

True positive variants are those variants from the test set that are also present in the truth set, while false positive are the ones that aren't present in the truth set. On the other hand, false negatives are those variants present in the truth set, but missing from the test set. Precision, in this particular context, is fraction of true positive variants in the test set, while recall is fraction of true positive variants in the truth set. F-measure is the harmonic mean of precision and recall.

3.1 Simple algorithm

The most simple way of comparing VCF files would be to compare only chromosome names in which variants are found, their positions within chromosomes, as well as reference and alt alleles.

To read VCF `pysam` Python module [5] will be used, specifically its `VariantFile` class.

[1]:
```
import pysam

truthSet = pysam.VariantFile("vcf_files/gatk_result.vcf")
testSet = pysam.VariantFile("vcf_files/freebayes_result.freebayes.vcf")
```

The number of variants in each file can now be obtained.

[2]:
```
truthSet.reset()
testSet.reset()

truthCount = len(list(truthSet))
testCount = len(list(testSet))

print(f"Number of variants in truth set: {truthCount}")
print(f"Number of variants in test set: {testCount}")
```

```
Number of variants in truth set: 70181
Number of variants in test set: 77620
```

Now is the time to check if variants in files are in order or not. But, before that, since chromosome name is a string, it needs to be transformed into integer, which is comparable using lesser and greater than. Easiest way to do this is using dictionaries.

[3]:
```
chromDict = {
    "1": 1, "2": 2, "3": 3, "4": 4, "5": 5, "6": 6, "7": 7, "8": 8, "9": 9, "10":
 ↪ 10,
    "11": 11, "12": 12, "13": 13, "14": 14, "15": 15, "16": 16, "17": 17, "18":␣
 ↪18,
    "19": 19, "20": 20, "21": 21, "22": 22, "X": 23, "Y": 24, "MT": 25
}

truthTest = True
testTest = True

truthSet.reset()
testSet.reset()

prevRecord = next(truthSet)

for record in truthSet:
    if chromDict[prevRecord.chrom] > chromDict[record.chrom] or␣
 ↪(chromDict[prevRecord.chrom] == chromDict[record.chrom] and prevRecord.pos >␣
 ↪record.pos):
        truthTest = False
        break

    prevRecord = record
```

```
prevRecord = next(testSet)

for record in testSet:
    if chromDict[prevRecord.chrom] > chromDict[record.chrom] or␣
 ↪(chromDict[prevRecord.chrom] == chromDict[record.chrom] and prevRecord.pos >␣
 ↪record.pos):
        testTest = False
        break

    prevRecord = record

print("Truth set: " + "IN ORDER" if (truthTest) else "NOT IN ORDER")
print("Test set: " + "IN ORDER" if (testTest) else "NOT IN ORDER")
```

```
Truth set: IN ORDER
Test set: IN ORDER
```

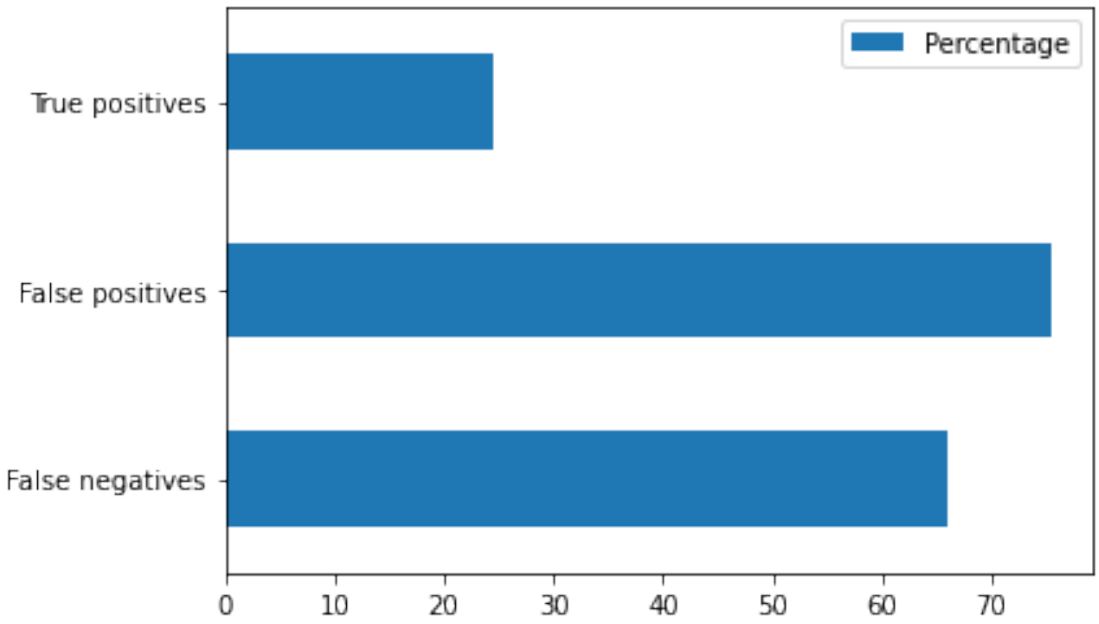
Knowing that both files are in order greatly reduces analysis time, since only one pass trough both of the files is required. It is enough to look for true positives, since number false positives and false negatives can be calculated by subtracting number of true positives from total number of variants in test set and truth set, respectively. Result can be visualized using `pandas` Python module [6].

[4]:
```
import pandas
import matplotlib

truePositives = 0

truthSet.reset()
testSet.reset()

testRecord = next(testSet)

for truthRecord in truthSet:
    try:
        if chromDict[truthRecord.chrom] == chromDict[testRecord.chrom]:
            if truthRecord.pos == testRecord.pos:
                if truthRecord.ref == testRecord.ref and truthRecord.alts ==␣
 ↪testRecord.alts:
                    truePositives += 1
                testRecord = next(testSet)
            elif truthRecord.pos > testRecord.pos:
                while chromDict[truthRecord.chrom] == chromDict[testRecord.
 ↪chrom] and truthRecord.pos > testRecord.pos:
                    testRecord = next(testSet)
        elif chromDict[truthRecord.chrom] > chromDict[testRecord.chrom]:
            while chromDict[truthRecord.chrom] > chromDict[testRecord.chrom]:
                testRecord = next(testSet)
```

```
    except:
        break

truthSet.close()
testSet.close()

falsePositives = testCount - truePositives
falseNegatives = truthCount - truePositives

def outputData(truePositives, falsePositives, falseNegatives, testCount):
    precision = truePositives / (truePositives + falsePositives)
    recall = truePositives / (truePositives + falseNegatives)
    fMeasure = 2 * precision * recall / (precision + recall)

    dataFrame1 = pandas.DataFrame([truePositives, falsePositives,␣
 ↪falseNegatives], ["True positives", "False positives", "False negatives"],␣
 ↪[""])

    dataFrame2 = pandas.DataFrame([precision, recall, fMeasure], ["Precision",␣
 ↪"Recall", "F-measure"], [""])

    dataFrame3 = pandas.DataFrame([falseNegatives / testCount * 100,␣
 ↪falsePositives / testCount * 100, truePositives/ testCount * 100], ["False␣
 ↪negatives", "False positives", "True positives"], ["Percentage"])

    display(dataFrame1.T)
    display(dataFrame2.T)
    dataFrame3.plot.barh();

outputData(truePositives, falsePositives, falseNegatives, testCount)
```

	True positives	False positives	False negatives
	18955	58665	51226

	Precision	Recall	F-measure
	0.244203	0.270087	0.256494



It can be seen that there is a big discrepancy between the results. This is because proposed analysis is too basic and doesn't take a lot of things into consideration. For example, these tools might format same variants in a different way, which is exactly what happened with variants on positions 889158 and 889159 in chromosome 1.

Truth set:

```
1   889158  .   G   C
1   889159  .   A   C
```

Test set:

```
1   889158  .   GA  CC
```

Fortunately, there are available tools that execute more advanced analysis.

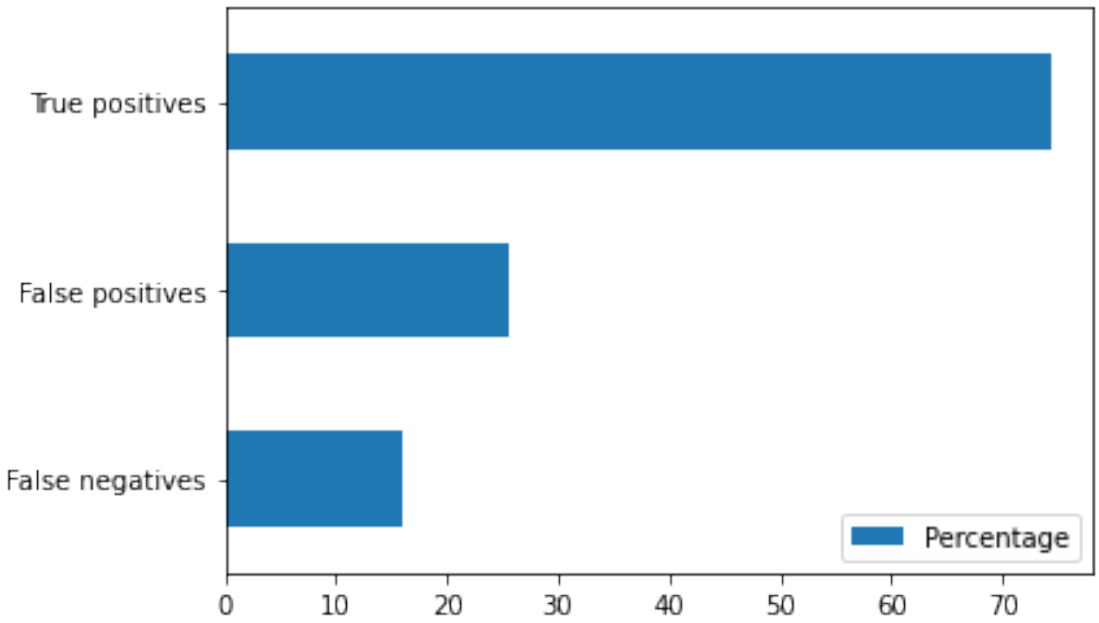
3.2 BCFtools

BCFtools [7] is a program for variant calling and manipulating VCF and BCF files. Of special interest here is `bcftools isec` command, which can create intersections, unions and complements of VCF files. [8] As arguments, it requires two VCF files to compare and returns four VCF files that represent records private to first file, records private second file, records from first file shared by both and records from second file shared by both, as well as a TXT file that explains the contents of aforementioned VCF files. Therefore, only thing that needs to be done to analyze the results is to count records in these files.

[5]:
```
file0 = pysam.VariantFile("bcftools/0000.vcf")
file1 = pysam.VariantFile("bcftools/0001.vcf")
file2 = pysam.VariantFile("bcftools/0002.vcf")
```

```
truePositivesBCF = len(list(file2))
falsePositivesBCF = len(list(file0))
falseNegativesBCF = len(list(file1))

file0.close()
file1.close()
file2.close()

outputData(truePositivesBCF, falsePositivesBCF, falseNegativesBCF, testCount)
```

```
True positives  False positives  False negatives
         57835            19785            12346

Precision    Recall  F-measure
 0.745104  0.824083   0.782606
```



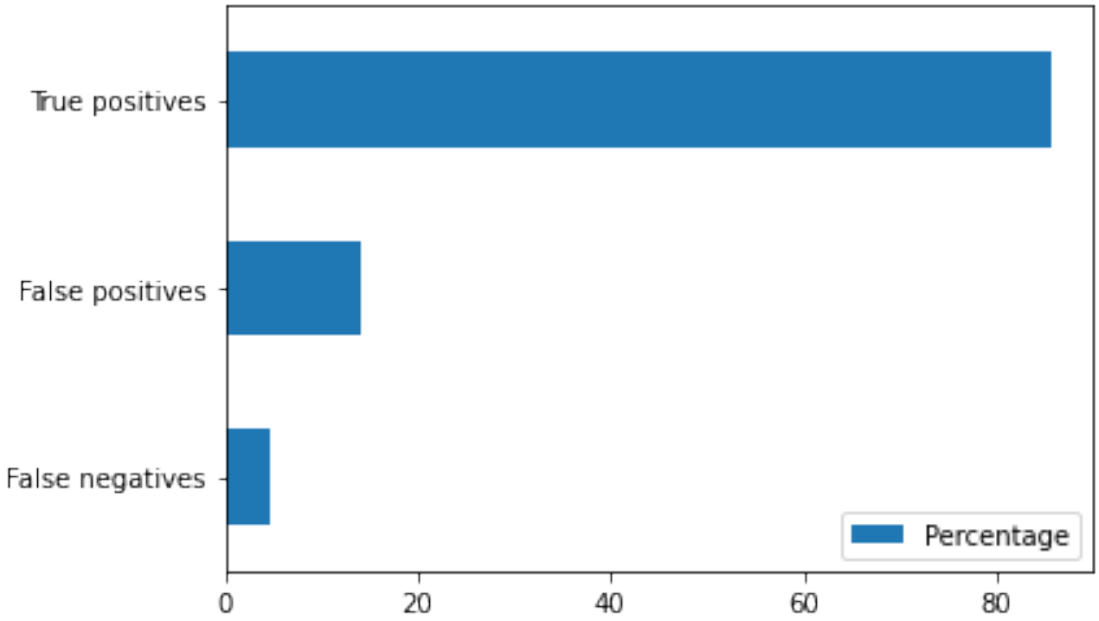
These results show that there actually is a lot more correlation between truth and test set than what simple algorithm detected. Precision of 0.75 is reasonably good, bot not amazing, while recall is a bit better at 0.82.

3.3 BEDTools

BEDTools [9] is a set of tools that can be used for a wide range of genomics analysis tasks. Specific tool required here is `bedtools intersect`, which takes two VCF files and returns their intersection. Although resulting file has VCF extension, it only contains intersecting variants and no header, so it cannot be read using `pysam.VariantFile`. It should, rather, be read as normal TXT file from which number of lines needs to be extracted.

[6]:
```
file = open("bedtools/gatk_result.intersect.freebayes_result.freebayes.vcf")

truePositivesBED = len(file.readlines())
falsePositivesBED = testCount - truePositivesBED
falseNegativesBED = truthCount - truePositivesBED

file.close()

outputData(truePositivesBED, falsePositivesBED, falseNegativesBED, testCount)
```

```
True positives  False positives  False negatives
         66587            11033             3594

Precision   Recall  F-measure
 0.857859  0.94879   0.901036
```



This tool performed slightly better than the last one, managing to find more correlations and get precision up to 0.86, which is pretty good, and recall up to 0.95.

3.4 VBT

VBT (Variant Benchmarking Tools) provides a set of tools that is used for aligner/variant calling benchmarking. [10] Its tool for variant comparison is useful for this analysis. It takes three arguments, baseline and called VCF, as well as reference FASTA file, and returns four VCF files, which contain false negative variants, false positive variants, true positive variants in baseline VCF and true positive variants in called VCF, as well as TXT file which is a summary of analysis and contains number of true positives in called VCF, true positives in baseline VCF, false positives, false

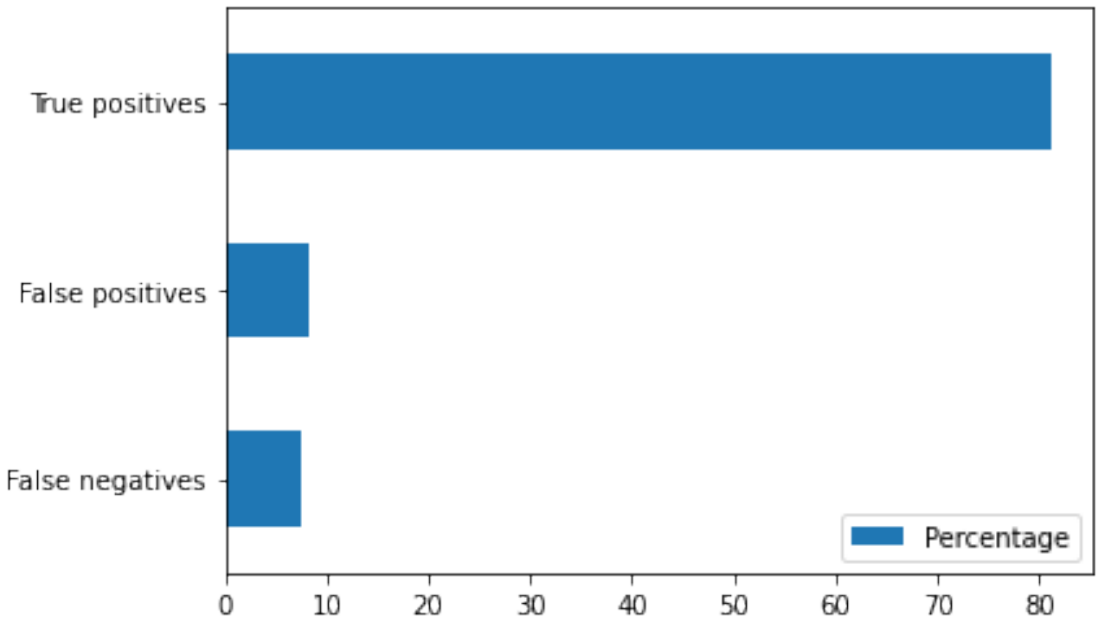
negatives, as well as calculated precision, recall and F-measure. This data is calculated for each chromosome individually as well as the entire genome.

Results should be, therefore, simply read from the file.

[7]:
```
truePositivesVBT = 63096
falsePositivesVBT = 6486
falseNegativesVBT = 5735

precisionVBT = 0.9068
recallVBT = 0.9182
fMeasureVBT = 0.9125

dataFrame1 = pandas.DataFrame([truePositivesVBT, falsePositivesVBT,␣
↪falseNegativesVBT], ["True positives", "False positives", "False negatives"],␣
↪[""])

dataFrame2 = pandas.DataFrame([precisionVBT, recallVBT, fMeasureVBT],␣
↪["Precision", "Recall", "F-measure"], [""])

dataFrame3 = pandas.DataFrame([falseNegativesVBT / testCount * 100,␣
↪falsePositivesVBT / testCount * 100, truePositivesVBT/ testCount * 100],␣
↪["False negatives", "False positives", "True positives"], ["Percentage"])

display(dataFrame1.T)
display(dataFrame2.T)
dataFrame3.plot.barh();
```

	True positives	False positives	False negatives
	63096	6486	5735

	Precision	Recall	F-measure
	0.9068	0.9182	0.9125



This tool gave the highest precision value of all tools we used, and slightly lower recall than the last one. This may be due to the fact that this tool, which is based of `vcfeval` [11], takes reference file into consideration while performing analysis.

It should be noted that recall and F-measure value would be slightly different if they were to be calculated the same way they were calculated before because this time tool recognizes that there can be a different number of records for true positive variants in files that are being compared.

4 References

[1] https://gatk.broadinstitute.org/hc/en-us

[2] https://github.com/freebayes/freebayes

[3] http://samtools.github.io/hts-specs/VCFv4.2.pdf

[4] https://en.wikipedia.org/wiki/Precision_and_recall

[5] https://github.com/pysam-developers/pysam

[6] https://pandas.pydata.org/

[7] http://samtools.github.io/bcftools/

[8] http://samtools.github.io/bcftools/howtos/index.html

[9] https://github.com/arq5x/bedtools2

[10] https://github.com/sbg/VBT-TrioAnalysis

[11] https://github.com/RealTimeGenomics/rtg-tools