**Response to Reviewers**

We would like to thank the editor and reviewers for their comments.

**VariantSpark: Population Scale Clustering of Genotype Information**

Aidan R O’Brien, Neil Saunders, Fabian A. Buske, Denis C Bauer

The reviewers’ comments and suggestions are boxed in gray and itemized below, followed by our responses and a description of the consequent revisions to the paper. Text from the manuscript or supplemental materials is italic and changes are “quoted in red”. Please note that in our responses we have used the abbreviations ‘pg’ and ‘para’ for page and paragraph, respectively.

**Reviewer: 1**

Figure 2 and 3 do not add much to the paper as they are poorly labeled and not easily interpretable.

We updated the figures captions to increase the interpretability of the figures

Figure 2

Schematic overview of VariantSpark. The image shows the flow from the input VCF file to the machine learning library and onto the visualization. It highlights the differences between the Hadoop and Spark implementations for converting data in VCF format to a data structure readable by Mahout and MLlib, respectively.

Figure3

Visualisation of VariantSpark predicted clusters. The figure shows the four clusters predicted for the 1000 Genomes data. Individuals from the super-populations AFR, AMR and EAS are accurately grouped into distinct clusters. The fourth cluster contains predominantly EUR + AMR individuals potentially accurately reflecting migrational backgrounds.

We also updated Figure 1 to 1) added binary-conversion for ADAM and 2) corrected labelling errors 3) capture the improved runtime for R

A better description of new features (in addition to a reasonable speed up) need to be discussed

We added the following sentence to the conclusion to address this point:

Utilising MLlib as well as Spark.ML will enable supervised machine learning applications to e.g. identify variants that jointly interact with phenotypes as well as include electronic health record in addition to the genomic feature vector to e.g. capture medical history as well as predispositions for diagnosis and treatment decisions.

**Reviewer 2:**

I wasn't able to find the reference implementations in R and Python.

We update the method section as follows

We also include the R and Python source file as supplementary material. Furthermore, we improved the R implementation to decrease the runtime from 75m to 42m, however this does not change the overall findings in the paper.

R Implementation

For our R implementation, we utilize read.table to read in a VCF file as a tab- delimited document and remove the first nine columns that do not contain variants. In the supplementary material we benchmark this against an approach reading the VCF file using the ‘VariantAnnotation’ package, which causes inefficiencies for the subsequent data manipulation tasks and is hence overall slower.

As with our VariantSpark pre-processing, we convert the strings that repre- sent each allele to a numeric value. This process is identical for both of our R approaches, and essentially consists of applying our Hamming function to the dataframe with sapply. We then transpose the matrix with t(vcfMatrix), which results in a data-structure where each row represents an individual. We convert the matrix to a big.matrix object, as required by the k-means algorithm from the ‘biganalytics’ package (https://cran.r-project.org/web/packages/ biganalytics/index.html), and then call bigkmeans with the big.matrix object and the required number of clusters as arguments.

Python Implementation

Our Python implementation reads in lines from a VCF file as tab-separated values using DataFrame.from csv, and stores the data in a pandas DataFrame (http://pandas.pydata.org). The column headings are the individual IDs and the row headings are the allele locations. We the first 9 columns and convert the remaining allele strings to numeric values. We convert the DataFrame to a matrix with .as\_matrix() and cluster the matrix using sci-kit learn (http://scikit- learn.org/stable/).

As a baseline for performance comparisons it would appear more natural (to me), to use a C++ implementation, which would actually be very easy to do using the bcftools API to read VCF/BCF and mlpack for clustering.

We investigated bcftools and find it to be inferior compared to native BASH functions for parsing the VCF file. We further investigated mlpack for the kmeans clustering and find it not suitable as it does not support multithreading. It is hence not comparable to the other methods, which all utilize parallelization for the kmeans step including R and python. We did not find another machine learning library in C++, which can be started from command line, therefore presenting a solution in C++ would require a de-novo implementation and is hence outside the scope of this paper.

Scalability is an issue and will certainly become ever more important, especially in the light of projects like Genomics England and the Precision Medicine Initiative. So exploring the use concepts like Spark is certainly welcome. However, VariantSpark caters to only one very specific use case, rather than providing a platform to address many problems one faces. In order for it to become widely adopted, it would need to have a wider scope (in my opinion).

We agree that the reviewed version of the paper focuses on only one application we hence elaborated on the options of using the other machine learning algorithms that are enabled by VariantSpark (See Reviewer1 comment 2)

R and Python do have specific packages to parse VCFs (Bioconductor/VariantAnnotation and pysam). I wonder whether using these would result in better performance.

We tested VariantAnnotation but find it does not speed-up the data handling. We added the following sentence:

Page 8, para 1

In the supplementary material we benchmark this against an approach reading the VCF file using the `VariantAnnotation' package, which causes inefficiencies for the subsequent data manipulation tasks and is hence overall slower.

Supplemental

COPY SUPPLEMENT

Generally, it is difficult for a specialized library to offer a more efficient way for generic data manipulation tasks than the built in functions that are designed for this purpose. We therefore expect to get a similar result for pysam, especially so since “The VCF/BCF API is preliminary and incomplete.” (<http://pysam.readthedocs.org/en/latest/usage.html)>. The dramatic improvement we observe using VariantSpark comes from parallelizing the tasks which neither VariantAnnotation nor pysam offers.

To demonstrate scalability to today's data sets, using the present 1000 Genomes release (phase 3) would be appropriate.

We agree with the reviewer and have included the phase 3 data in our paper.

Page 5, para 3

To further demonstrate the scalability of VariantSpark, we also cluster the 1000 Genomes Project phase 3 data, which contains 3000 individuals from 5 super-populations and as a result has over 80 Million variants. The uncompressed size of the phase 3 files is 770GB compared to the 161GB of the phase 1 dataset. VariantSpark successfully completes the clustering in just 27 hours (see Table 2) with an ARI of 0.82.