Permutect: An Elegant Method for Somatic and Germline Variant Calling

David Benjamin[1,✉], Ethan Benjamin[3], Lee Lichtenstein[1], Juan Gallegos[2], Sachet Shukla[2], Julian Gascoyne[2], Mehrtash Babadi[1], and Samuel Friedman[1]

[1]The Broad Institute
[2]MD Anderson
[3]Etsy

We present Permutect (Permutation-Invariant Mutect), the successor to Mutect2. Permutect combines a novel machine learning model of technical error with a probabilistic model of sample-specific biology, Permutect is able to genotype somatic and germline variants despite being trained using only germline data. By encoding permutation symmetry within a set of reads into its architecture, Permutect can obtain accurate results with very few parameters.

variant calling | somatic | Mutect | deep learning

Correspondence: ***davidben@broadinstitute.org***

Introduction

The first basic idea of Permutect is the simple observation that the set of sequencing reads supporting a variant exhibit permutation symmetry. Genotypes do not depend on which supporting read is considered the first, or seventh etc. Permutect is built from neural networks that respect this symmetry rather than ones that impose an arbitrary order on reads. Doing so permits a model with vastly reduced parameters with no loss of expressive power.

The second idea is to decouple variant calling into two independent parts. Detecting technical errors due to library preparation, sequencing, and alignment is a complex pattern detection problem well-suited to modeling with a neural network. In contrast, distinguishing different types of biological variation is a structured problem that can be modeled explicitly with fewer parameters and varies from sample to sample. Permutect combines a neural network model of technical error with a probabilistic genotyping model. Decoupling of models implies decoupling of training data: the Permutect neural network requires examples of technical error in training but does not require any somatic variants. We call the neural network model of technical error and the probabilistic biology model the artifact model and the posterior model, respectively.

Thanks to these two insights Permutect is not overfit to any particular sequencing technology, works for both somatic and germline calling, works on non-human data, can be trained on a very modest amount of data, and needs only germline data for training, regardless of the type of calling it needs to do.

In contrast to Permutect's decoupled approach, one could also train a single end-to-end neural network to take a set of reads as input and output a classification as somatic, germline, or non-mutated. Such networks need many examples of somatic variation to train on, which is problematic because labeled somatic truth data are extremely rare, and what little data exist are not always reliable. Furthermore, these truth data are particular to a specific sequencing technology.

There is a more pernicious problem: by far the strongest signal distinguishing technical error and somatic mutations from germline variation is the fraction of reads supporting the mutant allele. A fraction near 1/2 or 1 strongly suggests a germline variant, a low fraction is most likely an error, and something in between may be somatic. But what about very pure, monoclonal tumor samples with allele fractions near 1/2? Or cfDNA samples with very low somatic allele fractions? One could include many tumors with a wide variety of cell fractions in the training data, but in addition to complicating things this approach still imposes some arbitrary assumption on what allele fractions are likely. Unlike Permutect, the end-to-end approach needlessly bakes an implicit distribution of somatic allele fractions into the trained model and has no capacity to adapt to the actual distribution found in a particular sample.

In effect, Permutect decomposes variant calling into several questions[1]:

1. Are the mutant allele reads likely to be sequencing errors given the base qualities and number of reads?

2. Do the reads support a germline mutation?

3. Are the mutant allele reads a technical artifact?

Of these, the first two questions are exercises in probability and do not require a machine learning model with many parameters. The third question demands identifying patterns of error that are *not* modeled well by base qualities and an assumption of independent reads. It is difficult and well-suited to deep learning. However, it has nothing at all to do with identifying somatic variants! Instead of the two classes somatic and non-somatic, we train this deep learning model with data that either artifact or non-artifact. In practice we simply use germline variants as our non-artifact training data.

An attentive reader should object twice to this choice. First, germline variants are not the same as somatic vari-

[1] Permutect doesn't actually impose these as distinct criteria, rather different parts of the model contribute likelihoods from which posterior probabilities are computed jointly.

ants – certain variants are much more likely in some cancers, for example. This would be a compelling objection for an un-decomposed variant calling approach, but in Permutect the prior probabilities of different somatic variants are the responsibility of the probabilistic parts of the model, not of the deep learning artifact model. As a concrete example, suppose we have an A->T mutation within the context CGGCAGGCC. How likely an A->T mutation is to occur here is the purview of a few-parameter probabilistic model that Permutect fits anew for each sample. The artifact model is only concerned with the question: did a set of reads exhibiting the pattern CGGCTGGCC come from DNA with that same pattern, or did they arise from technical error and a DNA fragment with the CGGCAGGCC allele?

The second objection is that germline variants come in allele fractions that do not in general represent those found in tumors. To get around this problem, Permutect's deep learning model of technical artifacts ignores allele fractions by design[2]. Like the prior probability of different mutations, the characteristic allele fractions of somatic variants is a sample-specific property that doesn't require very many parameters. That being said, we are wary of hubris and, in addition to the fraction-agnostic model architecture we also downsample reads supporting germline variants in order to generate balanced training data where artifacts and non-artifacts occur with a similar range of read counts. That is, after downsampling artifact and non-artifact data are equally likely to have any given number of supporting reads.

Evaluations

Simulated Tumors. Reliable truth data for somatic mutations is extremely rare. One can generate simulated tumors with software, such as BAMSurgeon(**?**), that inserts substitutions and indels into real sequencing data. The virtues of this approach are first that the set of true mutations is known perfectly as it is specified by the program and secondly that all technical errors in the data are completely real, being the same unmodified errors that were present in the original BAM file. While it is true that the mutations generated in silico are not real, they *are* realistic, since a base changed in silico is indistinguishable from a base changed by mutation and sequenced correctly. The only unrealistic aspect is that the types of mutations – the relative amounts of different substitutions and indel sizes – do not reflect a real tumor. Nonetheless, as long as a good variety of mutations are simulated, data simulated in this way present basically the same challenges as real data.

Our simulated data includes the popular DREAM challenge WGS tumor-normal pairs(**?**), as well as our own synthetic data that we generated from Illumina HiseqX and Novaseq whole genomes sequenced at the Broad Institute. Whereas the DREAM challenge pairs split the reads from a single BAM file to form a “tumor" and “normal", our simulated tumor and normal come from independently-sequenced replicates of the same sample, adding a slight layer of extra realism. To model different tumor purities, we generated synthetic tumors at various somatic mutation allele frequencies.

SEQC2 sample. Because wet lab techniques for validating somatic mutations are very expensive, it is tempting to instead determine truth as the consensus between different variant calling pipelines. In our opinion this usually results in “truth" data of dubious quality. The errors of different algorithms are not independent, and therefore relying on a majority vote among them only gives an illusion of statistical power. For example, variants with low allele fraction are difficult for all algorithms; hence consensus callsets tend to omit such variants.

The SEQC2 Consortium(**?**) have meticulously created truth data for a breast cancer cell line that avoids the usual shortcomings of consensus data. Rather than simply combining several variant calling algorithms, they used multiple variant callers, multiple alignment algorithms, multiple sequencing technologies, and multiple sequencing centers. Additionally, they validated candidate mutations with extremely deep (greater than 2000x coverage) targeted sequencing. Finally, they validated a random subset of their results with wet lab techniques and showed that their high-confidence truth set is very reliable.

Methods

Labeling training data. Somewhat confusingly, one creates a Permutect dataset by running Mutect2[3] in a special Permutect dataset mode. If a germline truth VCF is available, any variant contained in that VCF is considered to be a non-artifact example[4] while any Mutect2 call missing from that VCF is considered an artifact. Otherwise, any variant that is common in the population and is supported by a sufficient fraction of reads is considered a non-artifact[5] and anything called by Mutect2 that is supported by a small fraction of reads and is rare in the population. Everything else is considered unlabeled data and is used in semi-supervised learning.

There are a few important subtleties to this. First, “called by Mutect2" means “called by Mutect2 with sufficient log-odds". The log-odds emitted by Mutect2 is the log likelihood ratio of a variant versus a sequencing error and is calculated using base qualities with the assumption of independent reads. A sequencing error in Permutect is not the same thing as a technical artifact! We define the latter as an error that is *not* described by base qualities and the assumption of independent reads. That is, a technical artifact depends on some hidden covariate that affects all reads. For example, if there is a single non-reference read with a base quality of 30, the log-odds from Mutect2 will be roughly 3. This is a sequencing error because it comes from nothing more nefarious than the possibility of error as described by the base quality. The

[2] More precisely, it ignores allele fractions except for calibrating its confidence.

[3] This is one good reason not to name it Mutect3.

[4] Recall that the Permutect neural network is responsible for classifying artifacts and non-artifacts, not somatic and/or germline variants.

[5] Because germline variants are plentiful these thresholds can be fairly strict without sacrificing too much training data.

base qualities “did their job", so to speak. In contrast, 10 incorrect reads with base qualities of 30, amounting to a log-odds of 30 or so are overwhelmingly unlikely to be due to independent errors each within a probability of 1 in 1000. It is much likelier that there is a common explanation, such as poor mappability, explaining all the reads' errors. We consider such a case to be a technical artifact. We do not need a neural network to learn how to translate from a phred-scaled base quality to an error probability, nor would we want our training data to drown in such trivial examples. Permutect's neural network artifact model is only concerned with errors that would fool an independent-reads model.

The next subtlety is the balance of training data. For most sequencing technologies, true technical artifacts (as opposed to sequencing error as described above) are much rarer than germline variants. Thus we discard most non-artifact examples, which come from downsampling germline variants, in order to get similar quantities of artifacts and non-artifacts[6] Out of the aforementioned fear that our model is not as count-agnostic as we hope we perform this balancing separately for each non-reference read count and variant type. For example, for each single-base insertion supported by 8 non-reference reads, the training dataset is balanced between artifacts and non-artifacts.

The final subtlety is the allele fraction criterion when no truth VCF is available to label training data. Marking calls with low (high) non-reference read counts as artifacts (non-artifacts) seems to betray the principles of Permutect by hard-coding the very count dependence we seek to eliminate. Fortunately, this is not the case. Because the architecture is designed to forget read counts and by virtue of our downsampling and dataset balancing, this process may introduce some errors into the training data, but as long as technical artifacts *generally* occur with low allele fractions the learned model will still learn to recognize the correct patterns, though perhaps with less confidence than it should have. Although there are errors in the training data, there is no systematic bias against variants with low allele fractions because Permutect is, by design, incapable of acquiring such a bias.

Preparing training data. Next, data must be converted into tensors for computation. We designed this with computational simplicity in mind; it is not optimized and we expect fancier approaches to be fruitful in the future. A single input[7] to the artifact model consists of i) a set of reference reads, and ii) a set of non-reference (henceforth referred to as “alt") reads, with each read encoded as a vector (one-dimensional tensor); iii) 21 bases of the reference and alt haplotypes, centered at the variant start position; and iv) a vector of additional information pertaining to the variant. We will now describe each of these in detail,

Reads are encoded as a vector of features: the read mapping quality, the base quality at the variant start, the binary read strand, the binary read pair orientation, the distance of the variant from the left and right ends of the read and of the fragment, the fragment length, and the number of substitution and indel mismatches of the read with respect to its assembled haplotype (not the reference haplotype!) at various distance thresholds from the variant. The reference and alt haplotypes, which are assembled in Mutect2[8], are each represented as five-channel one-hot encodings, with one channel each for A, C, G, T and a fifth deletion channel. An insertion in the alt haplotype relative to the reference is represented as a deletion in the reference allele. For example, for reference sequence ACGTGCA an insertion T -> TT is represented as a reference array ACGTDGC and an alt array ACGTTGC prior to one-hot encoding. (The somatic variant caller NeuSomatic(**?**) uses the same scheme.) The extra information (“info") vector consists of one element each for the number of STR repeats of repeat length 1, 2, 3, 4, 5, and 6 adjacent to the left and right of the variant, and a few Mutect2 annotations describing the complexity of the local assembly. Many of these are undoubtedly handcrafted, arbitrary, and redundant; empirically, though, they are effective.

Finally, the non-binary elements of the read and info vectors are quantile normalized. That is, an element in the eg 65th percentile of its corresponding feature is normalized to the point in the unit normal distribution at which the CDF is 0.65. Binary elements such as strand and read orientation are not normalized. After normalization, a single datum contains sets $\{\mathbf{a}_i\}$ and $\{\mathbf{r}_j\}$ of alt and ref reads, a two-dimensional (one dimension for the sequence, one for the channel) tensor $\mathbf{h}$ representing the ref and alt haplotypes, and a vector $\mathbf{e}$ for the extra info.

Artifact Model: bottom layers. Before the artifact model does anything exotic (to the extent that neural architecture acting on sets are exotic), it transforms each vector independently. In this paper MLP denotes a multi-layer perceptron, a stack of linear transformations separated by a non-linear activation function. We parameterize MLPs by their hidden and output dimensions, eg, $\mathrm{MLP}(10,20,10)$ has hidden layers of sizes 10 and 20 and 10-dimensional output. We also allow for residual connections, encoded by negative numbers. For example, suppose we have and MLP $\mathrm{Net} = \mathrm{MLP}(10,-2,5)$ acting on 7-dimensional input. Then Net contains four linear transformations $L_1 : \mathbb{R}_7 \to \mathbb{R}_{10}$, $L_2 : \mathbb{R}_{10} \to \mathbb{R}_{10}$, $L_3 : \mathbb{R}_{10} \to \mathbb{R}_{10}$, and $L_4 : \mathbb{R}_{10} \to \mathbb{R}_5$ as subnetworks, and the action on an input $\mathrm{x} \in \mathbb{R}_7$ is:

$$\mathbf{x}_1 = \psi(L_1(\mathbf{x})) \tag{1}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \psi(L_3(\psi(L_2(\mathbf{x}_1)))) \tag{2}$$

$$\mathrm{Net}(\mathrm{x}) = \psi(L_4(\mathbf{x}_2)), \tag{3}$$

where ψ in a nonlinear activation function.

At the bottom of the artifact model, each read vector is transformed independently by an MLP, the info is transformed by a different MLP, and the haplotype data is transformed by a one-dimensional, 10-channel (5 each for ref and

[6] In practice we retain by default 10 times as many non-artifacts but weight the loss function accordingly so that in effect the data are balanced.

[7] For the purposes of describing the model conceptually we ignore batching. In practice, a batch of candidate variants is the actual unit of processing.

[8] Recall that Permutect data are derived from Mutect2, run with the Permutect dataset option enabled.

alt stacked together) convolutional network. The output of each is a vector:

$$\mathbf{r}_i^{(1)} = \mathrm{MLP}_{\mathrm{ref}}(\mathbf{r}_i) \quad \textbf{(4)}$$

$$\mathbf{a}_i^{(1)} = \mathrm{MLP}_{\mathrm{alt}}(\mathbf{a}_i) \quad \textbf{(5)}$$

$$\mathbf{e}^{(1)} = \mathrm{MLP}_{\mathrm{info}}(\mathbf{e}) \quad \textbf{(6)}$$

$$\mathbf{s}^{(1)} = \mathrm{CNN}_{\mathrm{hap}}(\mathbf{s}). \quad \textbf{(7)}$$

The haplotype and info vectors are then concatenated onto each ref and alt read, independently, to obtain new sets of reads:

$$\mathbf{r}_i^{(2)} = \mathrm{Concat}\left(\mathbf{r}_i^{(1)}, \mathbf{e}^{(1)}, \mathbf{s}^{(1)}\right) \quad \textbf{(8)}$$

$$\mathbf{a}_i^{(2)} = \mathrm{Concat}\left(\mathbf{a}_i^{(1)}, \mathbf{e}^{(1)}, \mathbf{s}^{(1)}\right). \quad \textbf{(9)}$$

Artifact Model: set layers. Next, the transformed reads are passed through a symmetric version of a gated MLP(**?**) with cross-attention between the alt and ref reads. We modify the gated MLP to be permutation-invariant and count-agnostic by restricting attention between reads to act on averages. That is, each read is influenced only by the averages of functions of alt and ref reads. Letting LN denote layer norm, Lin denote a single-layer linear transformation, and ψ be some nonlinear activation function, one gated MLP block acts on input sets $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ as follows:

$$\mathbf{u}_{x,i} = \psi(\mathrm{Lin}_1(\mathrm{LN}(\mathbf{r}_i))) \quad \textbf{(10)}$$

$$\mathbf{v}_{x,i} = \mathrm{LN}(\psi(\mathrm{Lin}_2(\mathrm{LN}(\mathbf{r}_i)))) \quad \textbf{(11)}$$

$$\mathbf{u}_{y,i} = \psi(\mathrm{Lin}_3^{(n)}(\mathrm{LN}(\mathbf{a}_i^{(n)}))) \quad \textbf{(12)}$$

$$\mathbf{v}_{y,i} = \mathrm{LN}(\psi(\mathrm{Lin}_4(\mathrm{LN}(\mathbf{a}_i)))). \quad \textbf{(13)}$$

The mean fields of two of these transformations are used to make gate vectors, which multiply the other two transformed vectors and are added to the inputs residually to obtain the output:

$$\mathbf{g}_{x,i} = \alpha_x \mathbf{v}_{x,i} + \beta_{xx}\bar{\mathbf{v}}_x + \beta_{xy}\bar{\mathbf{v}}_y \quad \textbf{(14)}$$

$$\mathbf{g}_{y,i} = \alpha_y \mathbf{v}_{y,i} + \beta_{yy}\bar{\mathbf{v}}_y + \beta_{yx}\bar{\mathbf{v}}_x \quad \textbf{(15)}$$

$$\mathbf{x}_i' = \mathbf{x}_i + \mathbf{g}_{x,i} \odot \mathbf{u}_{x,i} \quad \textbf{(16)}$$

$$\mathbf{y}_i' = \mathbf{y}_i + \mathbf{g}_{y,i} \odot \mathbf{u}_{y,i}. \quad \textbf{(17)}$$

where $\bar{\mathbf{v}}_x$ etc denotes the average over all $\bar{\mathbf{v}}_{x,i}$ and $\odot$ is elementwise multiplication. This entire process defines a gated MLP block:

$$\left(\{\mathbf{x}_i'\}, \{\mathbf{y}_i'\}\right) \equiv \mathrm{gMLP}\left(\{\mathbf{x}_i\}, \{\mathbf{y}_i\}\right). \quad \textbf{(18)}$$

The artifact model composes several of these gated MLP blocks to obtain final transformed sets of ref and alt reads:

$$\left(\{\mathbf{r}_i'\}, \{\mathbf{a}_i'\}\right) = \mathrm{gMLP} \circ \cdots \circ \mathrm{gMLP}\left(\left\{\mathbf{r}_i^{(2)}\right\}, \left\{\mathbf{a}_i^{(2)}\right\}\right). \quad \textbf{(19)}$$

At this point we have transformed sets of ref and alt reads in a permutation-invariant manner, but we still need to aggregate them into a single summary feature vector. Simply taking an average would work, but we can be more expressive with a learned permutation-invariant set pooling function. We discard the transformed ref reads $\{\mathbf{r}_i'\}$ because at this point the transformed alt reads already contain a great deal of information from the original ref reads. Thus we only need to define a pooling function on the set of transformed alt read vectors. For input set $\{\mathbf{a}_i'\}$ we define the SetPool subnetwork as:

$$\mathbf{u}_i = \mathrm{MLP}_1(\mathbf{a}\prime_i) \quad \textbf{(20)}$$

$$\mathbf{v}_i = \mathrm{MLP}_2(\mathbf{a}_i') \quad \textbf{(21)}$$

$$\{\mathbf{w}_i\} = \mathrm{Softmax}\left(\{\mathbf{v}_i\}\right) \quad \textbf{(22)}$$

$$\mathrm{SetPool}\left(\{\mathbf{a}_i'\}\right) \equiv \mathrm{MLP}_3\left(\sum_i \mathbf{w}_i \odot \mathbf{v}_i\right), \quad \textbf{(23)}$$

where the softmax acts over the i index. Due to the softmax each feature column $w_{:,f}$ is normalized over reads: $\sum_i w_{i,f} = 1$. We can think of w_{if} as the influence read i has in deciding feature f. It is easy to see that this set pooling operator can realize an arithmetic mean by learning an identity mapping $\mathbf{u}_i = \mathbf{x}_i$ and a constant mapping $\mathbf{v}_i = 0$. It can learn to compute a featurewise maximum by scaling the elements of $\mathbf{v}_i$ by a large constant. Thus this set pooling is a flexible and permutation-invariant method for learning nonlinear voting schemes.

Next the artifact model applies a linear layer with one-dimensional output to obtain a logit representing the likelihood that the candidate variant is an artifact or not.

Calibration Model. Up to this point the use of mean fields ensures that Permutect is agnostic to the total count of ref and alt reads, which is intentional as discussed above. For somatic variant calling and other situations where the prior on variants of interest may be very small it is essential to train not just an accurate classifier but a well-calibrated one. Although it is antithetical to our purposes to use read counts to distinguish between artifacts and real mutations, it is perfectly acceptable to use read counts to calibrate the output. To emit a final calibrated logit Permutect transforms the alt and ref read counts into vectors via a Gaussian comb featurization:

$$\mathrm{GC}(n) \equiv \mathrm{Softmax}\left(-(n-1)^2, -(n-2)^2 \cdots - (n-20)^2\right) \quad \textbf{(24)}$$

It concatenates the read count featurizations with the uncalibrated logit and passes them through a final MLP with output dimension 1 corresponding to the calibrated logit:

$$\mathrm{logit} \to \mathrm{MLP}\left(\mathrm{Concat}\left(\mathrm{logit}, \mathrm{GC}(n_{\mathrm{ref}}), \mathrm{GC}(n_{\mathrm{alt}})\right)\right). \quad \textbf{(25)}$$

To prevent the calibration from overriding the original logit, we constrain this MLP to be monotonic in the logit. This can be achieved by constraining any coefficient multiplying the logit in the first linear layer, as well as all coefficients in subsequent layers, to be positive.

Training the artifact model

For labeled data the artifact model is trained with the usual cross-entropy loss function. Unlabeled data are given an entropy-minimization loss function that promotes clustering:

$$\mathrm{entropy}(\mathrm{logit}) = -p\log(p) - (1-p)\log(1-p), \qquad \textbf{(26)}$$

where $p = \sigma(\mathrm{logit})$ is the mapping from logits to probabilities via the sigmoid function.

The cross entropy loss is minimized when the output probabilities of the model match the actual artifact probabilities given the covariates. Since we train on balanced data containing equal amounts of artifacts and non-artifacts, our training data has balanced prior probabilities, and therefore the learned probabilities that a variant is an artifact given the input data is proportional to the likelihood of the input data given that the variant is an artifact. This observation, that a model trained on balanced data learns not just probabilities but likelihoods, enables the posterior model to use the output of the artifact model, as we shall see below.

ACKNOWLEDGEMENTS
We acknowledge Giles Hall for suggesting the use of memory-mapped datasets.

Bibliography

Supplementary Note 1: Something about something