GATK TUTORIAL :: Variant Callset Evaluation & Filtering

WORKSHEET

This document is the worksheet for the GATK Introduction to Variant Filtering and Evaluation tutorial. It is complemented by an appendix document (available separately) that contains a summary introduction to the scientific context of the tutorial, as well as a Materials and Methods section describing the software and datasets used in the hands-on tutorial.

GATK Hands-On Tutorial:
Introduction to Variant Callset Filtering and Evaluation with GATK

This GATK workshop tutorial session focuses on key steps for evaluating a variant callset and determining differences between hard filtering and filtering with VQSR.

Our main purpose in this tutorial is to show you key differences between hard-filtering methods and variant recalibration (VQSR). We will demonstrate the tradeoff between sensitivity and false discovery rate in your variant callset and will show you how to use R to plot annotations -- which is helpful to understand why some variants get filtered out by one method but not the other. Our ultimate goal is to enable you to use these techniques to filter and evaluate your own variant callset.
1 EVALUATING YOUR VARIANT CALLSET

We will use VariantEval to determine sensitivity of all three VCFs (unfiltered, hard filtered, and VQSR filtered).

From the figure above, we can see there are 9 different possible sets:

1) In GIAB only (FN)
2) In GIAB and failed VQSR only (FN)
3) In GIAB and failed both hard filtering and VQSR (FN)
4) In GIAB and failed hard filtering only (FN)
5) In GIAB and passed both hard filtering and VQSR (TP)
6) Not in GIAB and failed VQSR only (TN)
7) Not in GIAB and failed both hard filtering and VQSR (TN)
8) Not in GIAB and failed hard filtering only (TN)
9) Not in GIAB and passed both hard filtering and VQSR (FP)

1.1 Using VariantEval to Determine Sensitivity

First we are going to run VariantEval to determine the sensitivity of the unfiltered VCF, hard filtered VCF, and VQSR filtered VCF.
```bash
dir java -jar GenomeAnalysisTK.jar \
    -T VariantEval \
    -R ref/human_g1k_b37_20.fasta \
    -eval vcf5/NA12878.unfiltered.vcf \
    -comp truth_dataset/NA12878.GIAB.vcf \
    -o sandbox/Unfiltered.VariantEval.txt \
    -noEV \
    -EV ValidationReport
```

**Note:** The `-noEV` specifies that we do not want all standard evaluation modules that are normally output by VariantEval by default. We simply want to look at the sensitivity, and we can get that with the `ValidationReport` evaluation module.

To view the output, we can use the Linux command `more`

```bash
dir more sandbox/Unfiltered.VariantEval.txt
```

The output should look like this:

```
# ValidationReport
# ValidationReport comp eval none all 64193 68079 0 114 0 0 0.00 0.00 0.00 100.00 100.00
# ValidationReport comp eval known all 64193 68079 0 114 0 0 0.00 0.00 0.00 100.00
# ValidationReport comp eval novel all 64193 68079 0 114 0 0 0.00 0.00 0.00 100.00
```

The table is hard to read, but it is much better represented in RStudio. Open up RStudio, and in the top right box, click on Import Dataset.

Choose “From Text File” and click on Unfiltered.VariantEval.txt. You should see a table like this:
When you scroll to the right, you can see the TP, FP, FN, TN, and sensitivity. TP, FP, FN, TN stand for true positives, false positives, false negatives, and true negatives respectively. We will focus on sensitivity for this workshop. Remember:

\[
\text{Sensitivity} = \frac{TP}{TP + FN}
\]

We can see from our table TP = 64079 and FN = 114. Sensitivity = 64079 / (64079+114) = 99.82

Now, we will run VariantEval on the hard filtered VCF.

```
java -jar GenomeAnalysisTK.jar \
  -T VariantEval \n  -R ref/human_g1k_b37_20.fasta \n  -eval vcf$s/NA12878.hard.filtered.vcf \n  -comp truth_dataset/NA12878.GIAB.vcf \n  -o sandbox/Hard.Filtered.VariantEval.txt \n  -noEV \n  -EV ValidationReport
```

After looking in RStudio, you should see that the number of true positives is 64031 and the sensitivity is 99.75, slightly less than that of the unfiltered VCF.

Now, let's run VariantEval on the VQSR filtered VCF.

```
java -jar GenomeAnalysisTK.jar \
  -T VariantEval \n  -R ref/human_g1k_b37_20.fasta \n  -eval vcf$s/NA12878.VQSR.filtered.vcf \n  -comp truth_dataset/NA12878.GIAB.vcf \n  -o sandbox/VQSR.Filtered.VariantEval.txt \n  -noEV \n  -EV ValidationReport
```

Notice the number of true positives is 63272 and sensitivity has dropped to 98.57 in the VQSR filtered VCF.

Did you expect the sensitivity to drop with filtering? The reason the sensitivity drops with filtering is that filtering removes both false positives and true positives. However, we still recommend filtering because it will remove more false positives than it will remove true positives. It is ultimately up to you to determine how many true positives you are willing to remove while removing false positives.
1.2 Determining False Discovery Rate

Now that we have seen filtering can remove some true positives, we will see that it also removes many false positives. We will see that the False Discovery Rate is much lower for the VQSR filtered VCF than for the other two VCFs. Remember:

\[
\text{False Discovery Rate} = \frac{FP}{TP + FP}
\]

For the purposes of this workshop, we will assume that any variant in our callset that is not in the GIAB dataset is a false positive. To get the variants that are in our callset but not in the GIAB dataset, we will use SelectVariants.

1.2.1 Run SelectVariants

We will first run SelectVariants on all three VCFs. This will give us the variants that are in our callset but not in the GIAB VCF. The output VCF contains the false positives.

```bash
java -jar GenomeAnalysisTK.jar \
-T SelectVariants \n-R ref/human_g1k_b37_20.fasta \n-V vcf//NA12878.unfiltered.vcf \n--discordance_truth_dataset//NA12878.GIAB.vcf \n-o sandbox/Unfiltered.FalsePositives.vcf
```

```bash
java -jar GenomeAnalysisTK.jar \
-T SelectVariants \n-R ref/human_g1k_b37_20.fasta \n-V vcf//NA12878.hard.filtered.vcf \n--discordance_truth_dataset//NA12878.GIAB.vcf \n-o sandbox/Hard.Filtered.FalsePositives.vcf
```

```bash
java -jar GenomeAnalysisTK.jar \
-T SelectVariants \n-R ref/human_g1k_b37_20.fasta \n-V vcf//NA12878.VQSR.filtered.vcf \n--discordance_truth_dataset//NA12878.GIAB.vcf \n-o sandbox/VQSR.Filtered.FalsePositives.vcf
```
1.2.2 Run VariantsToTable

Now, we need to run VariantsToTable to get an accurate count of the number of false positives in each of our callsets. Because of the header lines and not each variant site being on one line in the VCF, we cannot count the number of lines in the VCF to get the number of false positives. So, we will simply use VariantsToTable to get the variant sites on one line each and count the number of lines.

```
java -jar GenomeAnalysisTK.jar \
-T VariantsToTable \ 
-R ref/human_g1k_b37_20.fasta \ 
-V sandbox/Unfiltered.FalsePositives.vcf \ 
-F CHROM \ 
-F POS \ 
-o sandbox/Unfiltered.FalsePositives.table
```

```
java -jar GenomeAnalysisTK.jar \
-T VariantsToTable \ 
-R ref/human_g1k_b37_20.fasta \ 
-V sandbox/Hard.Filtered.FalsePositives.vcf \ 
-F CHROM \ 
-F POS \ 
-o sandbox/Hard.Filtered.FalsePositives.table
```

```
java -jar GenomeAnalysisTK.jar \
-T VariantsToTable \ 
-R ref/human_g1k_b37_20.fasta \ 
-V sandbox/VQSR.Filtered.FalsePositives.vcf \ 
-F CHROM \ 
-F POS \ 
-o sandbox/VQSR.Filtered.FalsePositives.table
```

Now that we have the false positives from each callset in table format, we can simply count the number of lines in each table to determine the number of false positives.

```
w c -l sandbox/Unfiltered.FalsePositives.table
```

You should see that the number of false positives in the unfiltered callset is 26881.

```
w c -l sandbox/Hard.Filtered.FalsePositives.table
```

The number of false positives in the hard filtered callset is 21894.

```
w c -l sandbox/VQSR.Filtered.FalsePositives.table
```

You can see the number of false positives in the VQSR filtered callset is 11015, which is the least of all the callsets.
1.2.3 Calculating False Discovery Rate

False Discovery Rate = \( \frac{FP}{TP + FP} \)

Unfiltered Callset:

False Discovery Rate = \( \frac{26881}{64079 + 26881} = 0.296 \)

Hard Filtered Callset:

False Discovery Rate = \( \frac{21894}{64031 + 21894} = 0.255 \)

VQSR Filtered Callset:

False Discovery Rate = \( \frac{11015}{63272 + 11015} = 0.148 \)

Although the VQSR filtered callset has a lower sensitivity than the other two callsets, it also has a lower False Discovery Rate. It is up to you to determine which you would rather have, a higher sensitivity or a lower False Discovery Rate. A higher sensitivity allows for both more true positives and false positives. A lower False Discovery Rate gives you fewer false positives, but also fewer true positives.

2 USING R TO COMPARE ANNOTATION VALUES OF VARIANTS IN EACH SET

In this section, we will learn how to plot the annotation values of variants and determine why some filtering techniques may or may not have passed/failed them.

We will plot the annotations QD and FS in both 1-dimension and 2-dimensions. By plotting the annotations, we will see the differences between hard filtering and VQSR with respect to the thresholds they apply to filter the annotations.

2.1 Run CombineVariants

To find out which set each variant belongs to, we can use CombineVariants. CombineVariants has a way to annotate each site with which set the site belongs to. For example, if a site is in GIAB and failed hard filtering but passed VQSR, CombineVariants will annotate the site with \( \text{set=G-filterInH-V} \). The “filterIn” flag before the filtering method tells us the site failed the filtering method, hence it was “filtered” in the set.
To view the set annotated VCF, we can simply view the end of the VCF using the Linux command `tail`

```
tail sandbox/NA12878.Combined.vcf
```

You should see something like this:

```
20  62961318 .  G  A  468.16  SNP_Filter;VQSRTrancheSNP99.90to100.00  AC=2;A F=0.500;AN=4;BaseQRankSum=2.39;DP=66;FS=5.316;GQ_MEAN=176.33;GQ_STDDEV=187.71;MQ=37.72;MQ0=0;MQRankSum =-8.100e-02;NCC=0;OD=7.32;ReadPosRankSum=1.48;SOR=1.284;VQSL0D=-7.790e+00;culprit=MQ;set=filteredInAlg T:AD:DP:GQ:PL ./.  0/1:16,17:33:99:427,0,393  0/1:16,17:33:99:427,0,393
```

Notice position 20:62962891 is annotated with `set=H-filterInV`. This means the site is not in GIAB, but passed hard filtering and failed VQSR. In other words, the site is part of set 6 above.

### 2.2 Use VariantsToTable to make R readable table of annotations

```
java -jar GenomeAnalysisTK.jar \
  -T VariantsToTable \ 
  -R ref/human_g1k_b37_20.fasta \ 
  -V sandbox/NA12878.Combined.vcf \ 
  -o sandbox/NA12878.Combined.table \ 
  -F CHROM \ 
  -F POS \ 
  -F set \ 
  -F QD \ 
  -F FS \ 
  --allowMissingData \ 
  --showFiltered
```

This command allows us to extract the chromosome, position, set, QD value, and FS value to a table that R can read. --allowMissingData allows any sites where any of the requested annotations are not present to still
be output. --showFiltered is necessary because by default VariantsToTable does not output sites that are filtered by both hard filtering and VQSR.

If we look at the output table, we can see a nice list of ordered columns with the information we requested.

more sandbox/NA12878.Combined.table

You should see something like this:

```
<table>
<thead>
<tr>
<th>CHROM</th>
<th>POS</th>
<th>set</th>
<th>QD</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>61098</td>
<td>H-V</td>
<td>16.95</td>
<td>0.991</td>
</tr>
<tr>
<td>20</td>
<td>61795</td>
<td>H-V</td>
<td>22.18</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>63244</td>
<td>H-V</td>
<td>17.85</td>
<td>1.797</td>
</tr>
<tr>
<td>20</td>
<td>63799</td>
<td>H-V</td>
<td>19.63</td>
<td>8.413</td>
</tr>
<tr>
<td>20</td>
<td>65980</td>
<td>H-V</td>
<td>17.25</td>
<td>0.670</td>
</tr>
<tr>
<td>20</td>
<td>66370</td>
<td>H-V</td>
<td>18.02</td>
<td>4.629</td>
</tr>
<tr>
<td>20</td>
<td>66720</td>
<td>H-V</td>
<td>12.59</td>
<td>3.736</td>
</tr>
<tr>
<td>20</td>
<td>68749</td>
<td>Intersection</td>
<td>28.51</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>70980</td>
<td>Intersection</td>
<td>11.75</td>
<td>0.813</td>
</tr>
<tr>
<td>20</td>
<td>72892</td>
<td>Intersection</td>
<td>11.12</td>
<td>3.863</td>
</tr>
</tbody>
</table>
```

Notice the first few variants have set = H-V. H-V means the variant is not in GIAB, but passed both hard filtering and VQSR. In other words, it is part of set 9 above. Now we can use this table in RStudio to make plots of the two annotations.

### 2.3 Plot QD and FS Using R

Open up RStudio and click on Import Dataset in the top right box. Choose NA12878.Combined.table and click Import.

You will see NA12878.Combined.table in the top left.
Just as a sanity check, let’s check what possible sets the variants can be in.

- `unique(NA12878.Combined$set)`

This checks the possible unique values of set in our dataset. Note, there are 9 sets displayed that each correspond to one of our sets above. To understand the set names in R, we will link them to the 9 sets we described above:

<table>
<thead>
<tr>
<th>In GIAB only</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>In GIAB and failed VQSR only</td>
<td>G-H-filterInV</td>
</tr>
<tr>
<td>In GIAB and failed both hard filtering and VQSR</td>
<td>G-filterInH-filterInV</td>
</tr>
<tr>
<td>In GIAB and failed hard filtering only</td>
<td>G-filterInH-V</td>
</tr>
<tr>
<td>In GIAB and passed both hard filtering and VQSR</td>
<td>Intersection</td>
</tr>
<tr>
<td>Not in GIAB and failed VQSR only</td>
<td>H-filterInV</td>
</tr>
<tr>
<td>Not in GIAB and failed both hard filtering and VQSR</td>
<td>FilteredInAll</td>
</tr>
<tr>
<td>Not in GIAB and failed hard filtering only</td>
<td>filterInH-V</td>
</tr>
<tr>
<td>Not in GIAB and passed both hard filtering and VQSR</td>
<td>H-V</td>
</tr>
</tbody>
</table>

We will plot QD and FS for all the variants in each of these sets. Note: We cannot plot variants for the first set (in GIAB only) because the GIAB VCF does not contain annotation values for QD or FS.

### 2.4.1 Plot QD in 1-dimension

Let’s take a look at what the QD annotation values look like for the variants in our dataset.

First, we need to load ggplot2:

- `library(ggplot2)`

This allows us to use ggplot2.

- `QD.plot <- ggplot(NA12878.Combined, aes(x=QD)) + geom_density(alpha=0.2)`
- `QD.plot`

When you type the above command into the console, you should see a plot like this:
Notice the two peaks around 15 and 33. QD is the Qual score divided by depth at the site. Our hard filtering recommendations say to fail variants with QD less than 2. It certainly looks like most of the variants have QD value greater than 2.

Next, let’s take a look at the QD values for the different sets.

- QD.Sets.plot <- ggplot(NA12878.Combined, aes(x=QD, fill=set)) + geom_density(alpha=0.2)

You should see something like this:

Pretty cool, huh? But, it’s hard to differentiate that different sets, so let’s break them down into different categories of true positives and false positives.

To get the true positive variants, we will assume any variant in GIAB is a true positive.

- TruePositives <- rbind(NA12878.Combined[which(NA12878.Combined[,"set"]=='Intersection'),],
                          NA12878.Combined[which(NA12878.Combined[,"set"]=='G-H-filterInV'),],
                          NA12878.Combined[which(NA12878.Combined[,"set"]=='G-filterInH-V'),],
                          NA12878.Combined[which(NA12878.Combined[,"set"]=='G-filterInH-filterInV'),],
                          NA12878.Combined[which(NA12878.Combined[,"set"]=='G-filterInH-FiltfilterInV'),],
                          NA12878.Combined[which(NA12878.Combined[,"set"]=='G-filterInH-filterInV'),])

Next, we can plot the true positive QD values.
Notice the true positives have values all over the place, but the majority of them have QD greater than 10.

Next, let's plot the QD values for the false positives. To get the false positives, we will consider any variant not in GIAB to be a false positive.

```
FalsePositives <- rbind(NA12878.Combined[which(NA12878.Combined[,"set"]=="H-V"),],
                       NA12878.Combined[which(NA12878.Combined[,"set"]=="H-filterInV"),],
                       NA12878.Combined[which(NA12878.Combined[,"set"]=="filterInH-V"),],
                       NA12878.Combined[which(NA12878.Combined[,"set"]=="FilteredInAll"),])
```

You should see a plot like this:
Big difference from the true positive plot! Notice the false positives tend to have values clustered around 1-2. There are some variants that are false positives that have QD greater than 30 however. Those variants may have other annotation values that are not good.

2.4.2 Plot FS in 1-dimension

Next, we will plot FS values for all the variants in our callset.

```r
FS.plot <- ggplot(NA12878.Combined, aes(x=FS)) + geom_density(alpha=0.2)
FS.plot + scale_x_log10()
```

When you type the above command into the console, you should see a plot like this:

![Plot of FS values](image.png)

We transformed the x-axis into a log scale so we can better visualize the range of values. Notice most of the values are less than 10 and almost all are less than 100.

Next, let's take a look at the FS values for the different sets.

```r
FS.Sets.plot <- ggplot(NA12878.Combined, aes(x=FS,fill=set)) + geom_density(alpha=0.2)
FS.Sets.plot + scale_x_log10()
```

You should see something like this:
As cool as that is, let’s again break the graph into true positives and false positives.

```r
FS.TruePositives.plot <- ggplot(TruePositives, aes(x=FS, fill=set)) + geom_density(alpha=0.2)
FS.TruePositives.plot + scale_x_log10()
```

You should see a plot like this:

Notice all the variants now have FS values less than 100 and the majority of them have FS values less that 10.

Now let’s plot the FS values for the false positives.

```r
FS.FalsePositives.plot <- ggplot(FalsePositives, aes(x=FS, fill=set)) + geom_density(alpha=0.2)
FS.FalsePositives.plot + scale_x_log10()
```

You should see a plot like this:
Notice the large number of variants with FS value greater than 100 here.

### 2.4.3 Plot QD and FS values in 2-dimensions

Next, we will plot both QD and FS in the same plot.

```r
QD.FS.plot <- ggplot(NA12878.Combined,
   aes(x = QD, y = FS, color = set)) + geom_point(size = 2)
```

The plot should look like this:

You can play around with the point size. However, even in this figure, we can see the variants with FS values in the high range are all mostly false positives (FilteredInAll). The Intersection true positives (bright pink) have mostly QD values above 10 and low FS values close to 0.
2.4.4 Plot QD and FS for variants in GIAB that passed hard filtering but failed VQSR

Next, we will plot the QD and FS values for variants that are true positives from hard filtering, but false negatives from VQSR.

```
HardFiltering.TruePositives <- subset(NA12878.Combined,
    set="G-H-filterInV")
```

HardFiltering.TruePositives is a new dataset. If you type HardFiltering.TruePositives into the console, you will get a long list of variants in only the set G-H-filterInV.

Now, let’s plot the annotations!

```
HardFiltering.TruePositives.plot <- ggplot(HardFiltering.TruePositives, aes(x = QD, y = FS, color = set)) + geom_point(size = 4)
```

```
HardFiltering.TruePositives.plot
```

You should see a plot like this:

![Plot of QD and FS](image)

Notice these are the variants that are in the GIAB dataset, but they failed VQSR yet passed hard filtering. Remember, we only filter out variants with QD < 2.0 and FS > 60.0 in hard filtering. All of these variants have FS < 60.0 and QD > 2.0. Why would VQSR filter these out? Let’s look at position 20:56866928 in IGV.

Open IGV and load the `human_g1k_b37_20.fasta` reference in the top left corner.
Then, choose NA12878_subset_20.bam to load.

Now that the bam file is loaded, we need to focus on a region of interest. Let’s look at position 20:56866928.

When you click enter, you should see something like this:
Notice the many soft clipped bases and many mapping quality 0 reads in the region. It looks like VQSR did a good job of filtering this site out! Now, let's look at the site in the VCF.

```
  zgrep -w 56866928 vcfs/NA12878.VQSR.filtered.vcf
```

We see the culprit is MQ. However, the MQ is 49.44, and we only filter out MQ < 40.0 in hard filtering. This is why the site passed hard filtering but not VQSR. Looking in IGV shows us we probably would not consider this a good site and that VQSR is correct. Of course, the site is in GIAB, but GIAB can have false positives too!

### 2.4.5 Plot QD and FS for variants in GIAB that passed VQSR but failed hard filtering

Next, we will plot QD and FS for the variants that are in GIAB and failed hard filtering only.

```
  VQSR.TruePositives <- subset(NA12878.Combined, set="G-filterInH-V")
  VQSR.TruePositives.plot <- ggplot(VQSR.TruePositives, aes(x = QD, y = FS, color = set)) +
                             geom_point(size = 4)
  VQSR.TruePositives.plot
```

The plot should look like this:
There are only three variants that are in GIAB but passed VQSR only! Notice the extreme values of FS and QD for the 3 three variants. Let’s look at position 20:41242638 in IGV. You should see something like this:

This site looks real. As you scroll through, notice the many reads that have the A SNP. Now, let’s look at the site in the VCF.

VQSR has Passed the variant, but it states the culprit is FS. VQSR gives each site a culprit even if it passes. The culprit is the annotation that lowers the VQSLOD score the most. FS = 70.607. That is pretty high.
Please note, that although it is high, the Phred-scaled value of 70.607 means we are 99.99999% sure there is strand bias. However, that also means there is 0.000001% chance there is no strand bias! Looking in IGV, we can see it does not look like there is much strand bias at all. It looks like VQSR has done a good job here.

### 2.4.6 Plot QD and FS for variants not in GIAB that passed hard filtering but failed VQSR

Next, we will plot QD and FS for the variants that are not in GIAB and passed hard filtering only.

```r
HardFiltering.FalsePositives <- subset(NA12878.Combined, set="H-filterInV")
```

```r
HardFiltering.FalsePositives.plot <- ggplot(HardFiltering.FalsePositives, aes(x = QD, y = FS, color = set)) + geom_point(size = 4)
The plot should look like this:
```

![Plot QD and FS for variants not in GIAB that passed hard filtering but failed VQSR](image)

There are a lot of variants here! VQSR has done a good job of getting rid of a lot of false positives. Note, hard filtering did not remove these variants because they have QD > 2.0 and FS < 60.0. Let's take a look at position 20:19099339 in IGV. You should see something like this:
Note the many reads that have 0 mapping quality. But, even more interesting is the homopolymer run in the region. The TG SNPs are only on the reverse strand. And, right before them is a TTTTTGGTTTGT sequence. These homopolymer runs confuse sequencers, and it is very likely the sequencer simply put in a TG SNP where it did not belong.

Looking at the site in the VCF, we see the culprit is MQ. We have seen in IGV that indeed MQ is bad in the region, however, the homopolymer runs are also contributing to the bad call.

### 2.4.7 Plot QD and FS for variants not in GIAB that passed VQSR but failed hard filtering

Next, we will plot QD and FS for the variants that are not in GIAB dataset and passed VQSR but failed hard filtering.

```r
VQSR.FalsePositives <- subset(NA12878.Combined, set=="filterInH-V")
VQSR.FalsePositives.plot <- ggplot(VQSR.FalsePositives,
    aes(x = QD, y = FS, color = set)) + geom_point(size = 4)
```

The plot should look like this:
Here is a perfect example of hard filtering’s strict boundaries. Clearly, we can see the variants that failed have QD < 2.0 and FS >60.0. Let’s look at why VQSR would not filter out one of the sites with a bad QD. Position 20:62589194 in IGV looks like this:

Notice the many repeats in the region. If we look at the site in the VCF, we see it has passed VQSR. We cannot tell the exact reason why it passed VQSR, but we do know the site must have looked good when the other dimensions are taken into account. For sites like this, one thing you can do is experiment with different sensitivity settings in VQSR. You can check if this site is dropped in a less sensitive level. For example, you could change the sensitivity level to 99.2 from 99.3 and check if this variant still passes. The higher the threshold at which it no longer passes, the more questionable the variant call.
3 FINAL LESSONS LEARNED

- Inevitable tradeoff between Sensitivity and Specificity when filtering
- VQSR allows you to determine cutoffs based on sensitivity to a truth set, unlike hard filtering.
- Comparing your variant callset to a truth dataset can be very helpful -- if it’s appropriate.
- Using R to plot annotations can help you see why some variants were filtered out by one filtering technique and not another.
- When it’s not obvious why a variant passed or failed filtering, you can use IGV to visualize the supporting data and make a determination based on experience and the information we provide about what the annotations mean.