Genetic Variant Call Filtering

Nico Chaves

All relevant code for the project can be found at:
https://github.com/nmchaves/variant-call-filter

I. Background

When an individual's genome differs from a reference genome at some location, we say that the individual has a "genetic variant" at that location. When a sequencing pipeline finds evidence for a variant, it outputs a "variant call" describing the variant. Obtaining an accurate set of variant calls for an individual is crucial to clinical genomics. However, in practice, variant call sets tend to contain many false positives. In other words, the genome of interest often does not contain many of the called variants. For example, approximately 17% of the variant calls are false positives in the dataset I used for this project.

There are 2 approaches to improve the quality of a variant call set. The first approach involves improving the upstream processes. For example, one may sequence the genome at higher coverage, integrate multiple sequencing methods to improve robustness, and integrate the individual's pedigree information (i.e. sequence the individual's relatives and take into account inheritance constraints) [1]. However, this would be prohibitively expensive to perform for every genome we would like to sequence. A more practical approach toward improving the quality of the variant call set is to develop a binary classifier that filters out the false positives while retaining as many of the true positives as possible. I pursue this second approach in this project.

To build such a classifier, I use features associated with the variant calls. Features are also referred to as "variant annotations" in this context. The features are generated by upstream processes, and they include: depth, quality score, strand bias, and several others. Briefly, depth is the number of reads supporting the variant call. A quality score represents the probability that a base was called correctly. The quality score for a given variant increases as the number of reads supporting that variant increases. As a result, it is useful to consider quality score/depth as a feature to capture the "average" quality of the reads supporting a given variant. Strand bias in sequencing refers to when one strand of DNA is favored over the other; a high strand bias may be indicative of a sequencing error. One can read about these and other features in the "Annotation Modules" section on the Genome Analysis Toolkit website [2].

II. Related Work

A. Genome Analysis Toolkit

The Genome Analysis Toolkit (GATK) [3] is a set of software tools for processing genomics data. The GATK includes a variant filtering tool called the Variant Quality Score Recalibrator (VQSR). The name reflects the fact that VQSR assigns a recalibrated quality score to each variant call. This new quality score is used to filter out low quality variant calls.

VQSR uses the features associated with the variant calls (including the original quality score) to train a Gaussian mixture model (GMM). VQSR only trains the GMM on variants which are known to exist in humans – for example, those that are found in the HapMap dataset [4]. In general, called variants which are already known to exist are more likely to be true positives. Thus, the GMM attempts to capture the various ways in which true positive variants cluster in the feature space. Let Q' denote the recalibrated quality score of a given variant. Q' is the maximum probability density of the variant, where the max is taken over all of the Gaussians in the GMM.

After building the GMM, VQSR filters out the variants with low Q', i.e. those with low probability density under *all* of the Gaussians. In particular, the user specificies a certain sensitivity threshold, such as $T = 99\%$. Then VQSR chooses a threshold Q_{99} such that 99% of the variants that it trained on satisfy $Q' \geq Q_{99}$. Clearly, if the user specifies a lower sensitivity threshold T, then Q_T will increase and VQSR will have fewer true positives and fewer false positives.

III. Dataset

I used data contained in the GATK resource bundle [5]. In particular, I used variant calls from chromosome 20 of individual NA12878. Chromosome 20 contains approximately 63 million base pairs, or approximately 2% of the total number of base pairs in the human genome. This dataset was generated using Illumina HiSeq technology at relatively low coverage (roughly 15x). The dataset contains 86,843 variant calls, 83.4% of which are true positives according to the gold standard dataset.

A. Gold standard Dataset

The gold standard dataset was generated using multiple sequencing techniques, and it consists of 89,426 high confidence variant calls from NA12878's chromosome 20 [6]. By definition, we assume that these are all real variants. Furthermore, we assume that any true positive variant found in the non-gold-standard dataset can also be found in the gold standard dataset. In other words, we assume that the gold standard dataset has not failed to identify any true variants that the non-gold-standard dataset has identified.

Note that the gold standard dataset contains more variants than the non-gold-standard dataset. This means that even if we filter the non-gold-standard dataset perfectly, we will still fail to identify some of the variants in the individual. However, the goal of this project is simply to filter out the false positive variant calls, not to discover false negatives.

B. Preprocessing

The raw variant call dataset is in VCF format, which is difficult to process, so I converted it to a tab-separted file using the GATK's VariantsToTable tool. When using this tool, I split multi-allelic variant calls into separate variant calls. For example, a variant call with bases A and T would be split into one variant call with base A and another variant call with base T. Multi-allelic variant calls are difficult to process without splitting, because some of their features are tuples. Moreover, the dataset only has approximately 20 multi-allelic variant calls, so splitting them will not have much effect on the overall results.

I then removed features with nonnumeric values. Some features do not have values for all variants (which results in "Not a number" values), and other features are represented as strings. Note: in the future, I intend to embed the string features as categorical variables. After this step, the following features remained (the corresponding VCF annotation symbols are given in parentheses):

1) quality score (QUAL)
2) allele count (AC)
3) allele frequency (AF)
4) depth (DP)
5) strand bias as measured by Fisher score (FS)
6) maximum likelihood expectation for the allele counts (MLEAC)
7) maximum likelihood expectation for the allele frequency (MLEAF)
8) root mean square mapping quality (MQ)
9) quality score normalized by number of reads supporting the variant (QD)
10) strand bias as measured by symmetric odds ratio (SOR)

Note that the QUAL feature here is the original quality score, not the score obtained from using VQSR.

I also used 2 binary features: (1) whether the variant is in the dbSNP dataset and (2) whether the variant was treated as a positive training site by VQSR (for example, this feature would be "1" if the variant is found in HapMap). These two features capture whether or not the variant is "common". Note that 97.6% of the variants in the dataset are found in dbSNP and 75.4% are treated as positive training sites by VQSR.

As a final preprocessing step, I scaled the features (except for the binary features) to have zero mean and unit variance, which some algorithms require.

C. t-SNE Embedding

After preprocessing the data, I generated t-SNE embeddings of the variant call dataset, as shown in Figure 1. Note that the axes in the embeddings are unitless. The data points are colored according to whether they are considered true positives by the gold standard dataset. To reduce computation time, I performed each embedding on a randomly selected subset of the variants. Figure 1(a) was generated from 5,000 randomly selected variants (4,217 of which are true positives), while Figure 1(b) was generated using 500 randomly selected variants (398 of which are true positives).

We observe that some of the false positives cluster together in the embeddings. This is especially apparent in Figure 1(b). This suggests that we should be able to build a binary classifier that filters out some of the false positives. However, there are still a large number of false positives dispersed widely around both embeddings. This suggests that the data is not separable, and we should not expect to attain perfect classification.

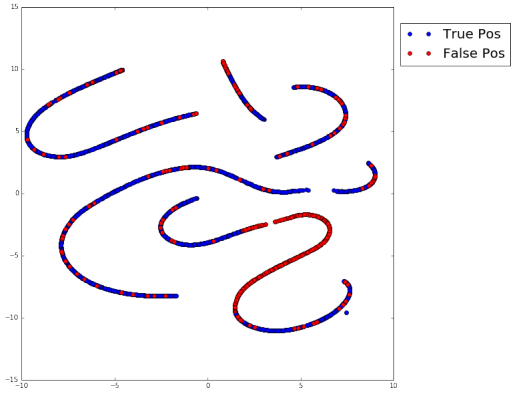
IV. Experiments

A. Supervised Learning

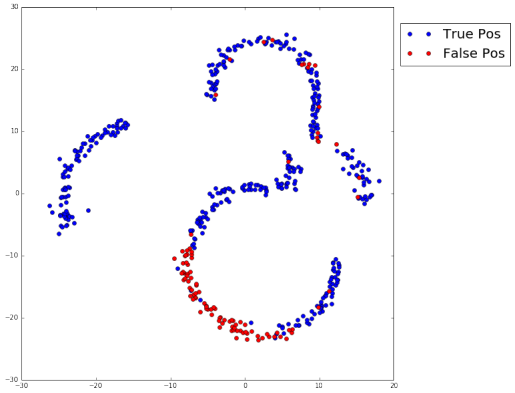
I used supervised learning algorithms to build a binary classifier that predicts whether a variant call is a true positive or not based on the features listed in the previous section. In particular, I used logistic regression, SVM (with Gaussian kernel), and random forest models. I used L2 regularization for both logistic regression and SVM.

I trained these supervised learning algorithms on a randomly selected 70% training set (60,790 variants) from chromosome 20 of individual NA12878. I used 10-fold cross validation and grid search to select the best hyperparameters for each model. I used accuracy as the metric for determining the best model during cross validation. I then evaluated the performance of the

Figure 1. t-SNE Embeddings



(a) 5,000 randomly selected variants. Perplexity=200.



(b) 500 randomly selected variants. Perplexity=30.

trained models on the held-out test set, i.e. the remaining 30% (26,053 variants) of the variant calls.

For random forest, I used 20 trees. I tried using up to 50 trees, but the validation error remained similar, while the execution time increased.

B. VQSR

To train VQSR, I used the 1000 Genomes project for the reference genome [7]. I also used the HapMap, dbSNP, and OMNI 2.5 datasets as described in the GATK best practices for training VQSR to filter SNP calls [8].

VQSR chose 75.4% of the variants (or 65,506 variants out of the total 86,843 variants) from the NA12878 chromosome 20 dataset for training the GMM.

Note that VQSR does not use a supervised learning approach, so it does not have a test set per se. However, I still evaluated it on the same test set as the supervised learning algorithms so that the methods can be compared on the same set of variants.

C. Execution Time

The execution times required by each approach are shown in Table I. The total execution time for VQSR consists of training the GMM and applying the filter. The total execution time of the supervised algorithms includes the time to select the model parameters (using 10-fold cross validation), the time to retrain the model on the training set, and the time to evaluate the model on the test set. Note that one could reduce the execution time of the supervised algorithms by using 5-fold cross validation or restricting the hyperparameter searches to a narrower range. However, both of these approaches may reduce the performance.

Table I
EXECUTION TIME OF EACH ALGORITHM

	Total Execution Time (seconds)
VQSR	767.7
Logistic Reg	94.7
SVM	1421.7
Rand Forest	227.4

Finally, note that VQSR must be trained on each new dataset it encounters, whereas the supervised learning algorithms only need to be trained once for a given sequencing technology and processing pipeline (at least in principle). In the future, I plan to evaluate whether a trained model can actually be applied to a different dataset (e.g. a variant call set from a different individual, but using the same sequencing technology and processing pipeline).

V. RESULTS

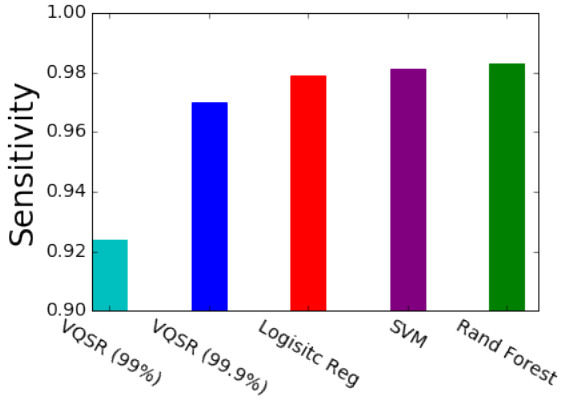
Figure 2 summarizes the performance of each approach described in the previous section. Figure 2 includes both the sensitivity (true positive rate) and specificity (1 - false positive rate) for each approach on the held-out test set. The results are also summarized in Table II, which includes the accuracy and F1 score of each approach.

VQSR (99%) refers to VQSR with a sensitivity threshold of 99%, and similarly for VQSR (99.9%). In other words, VQSR (99%) means that VQSR sets its threshold Q_{99} such that 99% of the variants it trained on are classified as true positives. This does *not* necessarily mean that VQSR (99%) will achieve 99% sensitivity overall. The 99% and 99.9% thresholds are automatically generated when following the GATK's best practices.

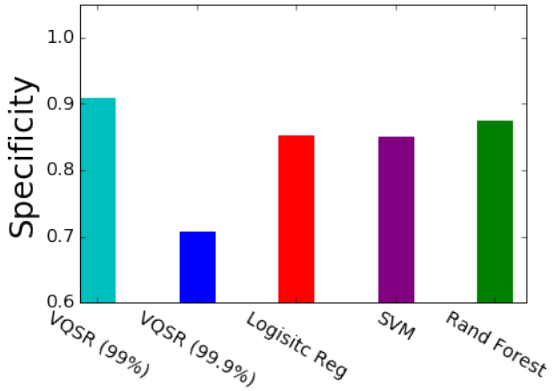
It is not surprising that the specificity is lower than the sensitivity in every approach. As shown in Figure 1, some of the false positives are spread among relatively dense regions of true positives. Since I optimized the supervised algorithms for accuracy, they likely incurred some false positives in order to correctly classify a larger number of true positives nearby.

Random forest has the best performance in terms of sensitivity, accuracy, and F1 score. VQSR (99%) has the

Figure 2. Performance Comparison on Test Set



(a) Note that the **y axis begins at 0.9** to help accentuate the differences.



(b) Note that the **y axis begins at 0.6** to help accentuate the differences.

Table II
PERFORMANCE COMPARISON ON TEST SET

	Sensitivity	Specificity	Accuracy	F1
VQSR (99%)	0.923	**0.903**	91.9%	0.923
VQSR (99.9%)	0.969	0.710	92.6%	0.923
Logistic Reg	0.978	0.852	95.8%	0.957
SVM	0.981	0.851	96.0%	0.959
Rand Forest	**0.983**	0.875	**96.5%**	**0.965**

Note: I would have reported the AUC score of each approach, but some of VQSR's recalibrated quality scores are reported as NaN values. I plan to resolve this issue in my future work.

best specificity, but its sensitivity and accuracy were the lowest out of all of the approaches. Therefore, random forest has the best overall performance.

More importantly, the three supervised learning algorithms performed very similarly, and they generally all performed better than VQSR. SVM might have performed better if I searched over a larger parameter space during the training phase. However, SVM has 2 hyperparameters to select (the penalty parameter and the kernel parameter), so the search space for the optimal hyperparameters is larger than the other supervised algorithms, which have only one hyperparameter to select. (As mentioned previously, I fixed the number of trees in the random forest to 20, so the only hyperparameter to select was the splitting criterion: gini index or entropy). As shown in Table I, SVM already had the longest execution time, so I chose not to increase its hyperparameter search space.

VI. CONCLUSIONS AND FUTURE WORK

This project has shown that supervised learning may be preferable over VQSR for improving the quality of genetic variant call sets. For example, random forest had better overall performance than VQSR, and it took less than 1/3 the time to execute. More sophisticated supervised learning algorithms, such as deep neural networks, may attain even better performance. I intend to test this in the future.

Before testing any other supervised learning algorithms, however, I intend to evaluate the current models on different datasets, such as variant calls from different chromosomes of NA12878 and from different individuals (but using the same sequencing technology and processing steps). It is not immediately clear that a model trained on one variant call set will perform well on a different variant call set.

As mentioned previously, I removed some nonnumeric features, which I plan to embed as categorical features in the future. As the number of features grows, feature selection will become more important. I intend to experiment with principal component analysis and mutual information feature selection. I also plan to try using these derived features as input to VQSR. This set of features may outperform the set of features recommended in the VQSR best practices.

VII. ACKNOWLEDGEMENTS

I'd like to thank Professor Tsachy Weissman, Kedar Tatwawadi, and Yanjun Han for their feedback and guidance. I'd also like to thank Idoia Ochoa and Mikel Hernaez for suggesting this project topic and directing me toward useful resources. Their guidance was also very helpful throughout the course of the project.

REFERENCES

[1] Eberle, Michael A., et al. "A reference dataset of 5.4 million human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree." bioRxiv (2016): 055541.
[2] VCF Feature Descriptions https://software.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_gatk_tools_walkers_annotator_VariantAnnotator.php.
[3] McKenna, Aaron, et al. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data." Genome research 20.9 (2010): 1297-1303.
[4] Gibbs, Richard A., et al. "The international HapMap project." Nature 426.6968 (2003): 789-796.

[5] GATK Bundle https://software.broadinstitute.org/gatk/download/bundle
[6] Zook, Justin M., et al. "Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls." (2014).
[7] 1000 Genomes Project Consortium. "A global reference for human genetic variation." Nature 526.7571 (2015): 68-74.
[8] VQSR Best Practices https://software.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_gatk_tools_walkers_variantrecalibration_VariantRecalibrator.php