Research in Industrial Projects for Students

Sponsor

Beijing Genomics Institute

Final Report

Detecting Carcinogenic Somatic Mutations

Student Members

David Stevens (Project Manager)

Chak Pong Chung

Kaiqi Yang

Weicheng Ye

Academic Mentor

Mr. Hau Man Yeung

Sponsoring Mentors

Mr. Binghang Liu

August 08, 2014

This project was jointly supported by Beijing Genomics Institute and NSF Grant ⟨ID⟩(get NSF grant ID from IPAM front office).

Abstract

In this project, we detect the somatic SNPs in chromosome 1 of a breast cancer patient. The data is provided by Beijing Genomics Institute (BGI). We are given two sets of data from one patient: one set of data from the normal tissue and another set from the tumor tissue. Using the genotyping tools SAMtools and Genome Analysis Toolkit (GATK), we extract relevant sequencing information to form features for use in machine learning. We extract 62 features in total for each SNP position. Since the original data had some formatting problems and missing-data problems, we screen and impute the 62 features to fit our data for use in nine classification machine learning algorithms. We applied Principle Components Analysis to the data after imputation and screening and take the top 10 principle components.Then we used neural network,SVM(linear and RBF kernel),decision tree,ramdom forest,KNN,LDA and QDA to test our data. These 9 classifiers assign binary labels to each SNP position, representing somatic and non-somatic mutations and the summation of these labels is calculated. We assign a label of somatic to those SNPs whose summation label is greater than or equal to 8 and send these SNPs to an annotation website, SNPnexus for cross-reference against several gene functional and disease phenotype databases. We find somatic SNPs in three known breast cancer-related genes: PARP1,MAP1A,TACSTD2.

Keywords: breast cancer, somatic, chromosome 1, features, classifiers, machine learning

Acknowledgments

We have several individuals and organizations that we would like to thank for making this project possible:

- RIPS-HK is organized by HKUST and IPAM. We thank these organization for their work in making RIPS happen.
- Special thanks to the organizers Prof. Shingyu Leung, Dr. Albert Ku, Dr. Avery Ching and Dr. Chi Wai Yu.
- Thank you to BGI for sponsoring this project and providing the sequencing data.
- Our industrial mentor, Mr. Binghang Liu, was instrumental in aiding our understanding of the project and gave us a lot of background knowledge in the early stages of the program.
- Our academic mentor, Mr. Hau Man Yeung advised us throughout the entirety of the program and provided valuable guidance in the whole process.

Chapter 1

Background

1.1 A Few Words on Our Sponsor

Beijing Genomics Institute (BGI) is one of the world's premier genome sequencing centers. BGI's history is one of firsts. It was the first institution to use next-generation sequencing to assemble a human and non-human mammalian (Giant Panda) genome. It was the first center to sequence the genome of an Asian individual, as part of the Yan Huang Project.

BGI was created by Yang Huanming and others in November 1999 in Beijing as an independent (non-government) research institute. The institution was instrumental in the execution of the Human Genome Project as the representative of China.

In 2007, BGI relocated its headquarters in Shenzhen and became "the first citizen-managed, non-profit research institution in China". In 2010, BGI Shenzhen bought 128 sequencing machines to increase its sequencing power and is currently the world's largest genome sequencing center. BGI is currently a key sequencing center in the 1000 Genomes Project, which aims to catalogue human genetic variation by sequencing the genomes of at least one thousand humans from various ethnic backgrounds, as well as the 1000 Plant Genomes Project.

BGI aims to promote research collaboration and provide scientific support to scientists all over the world, contribute to the advancement of innovative biological research. BGI is dedicated to facilitating the development of beneficial applications to healthcare, agriculture, and the environment, and create better lives for people all over the world. [1]

1.1.1 Somatic Mutations

A somatic mutation is a mutation acquired over the course of an individual's lifetime. This is in contrast to germline mutations, which are present at birth. The difference between the two types of mutations and their relationship to paired normal-tumor samples are illustrated in Fig. 1.1 and Fig. 1.2, respectively.

A major part of an organism, such as an entire branch of a tree or a complete tissue layer of an animal, may carry the mutation; it may or may not be expressed visibly. These mutations can (but do not always) cause cancer or other diseases.

We are interested in the somatic mutations in the cancer genome, with the intent to locate the driver somatic mutations so that we may have better methods of cancer prevention, diagnosis, and treatment.

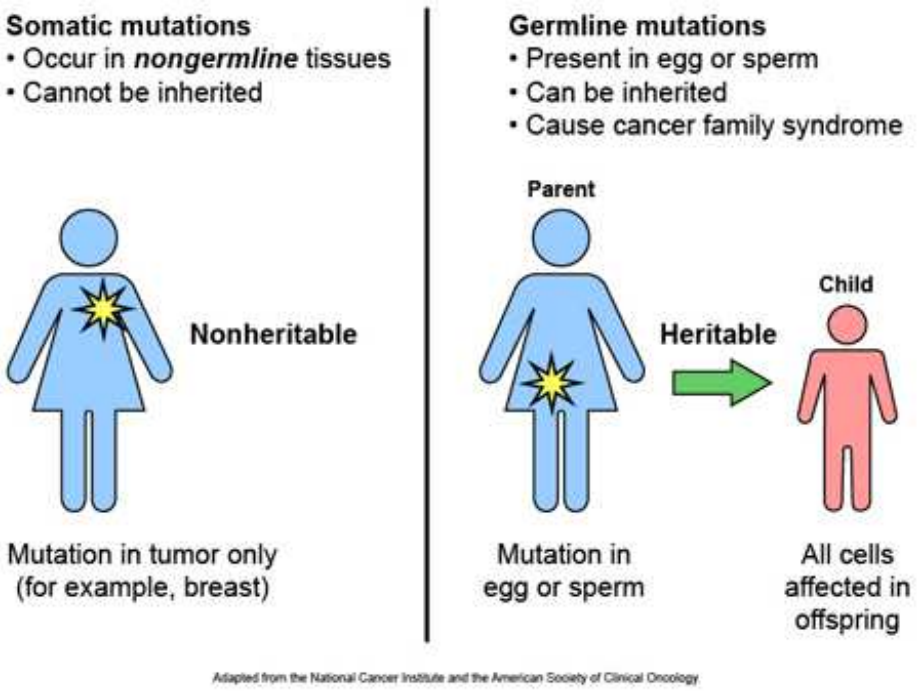


Figure 1.1: An illustration of the differences between germline and somatic mutations. Somatic mutations are non-heritable and occur in somatic tissues, whereas germ line mutations occur in germ line tissues and therefore may be inherited. (Source: `http://www.stjude.org/stjude/v/index.jsp?vgnextoid=5817b5b36e70c210VgnVCM1000001e0215acRCRD`)

1.1.2 Next-Generation Sequencing

Our genomic data was sequenced by Illumina's next-generation sequencing (NGS) technology, which utilizes a technique called sequencing by synthesis. NGS extends the traditional sequencing framework in a massively parallel fashion to sequence DNA with greatly increased speed and reduced cost. The rate of increase of NGS data output has exceeded even the rate predicted by Moore's law. We have seen an approximately 1000-fold rate increase in the four years between 2007 and 2011. In 2007, one sequencing run produced about a gig abase (Gb) of data, but by 2011 a single sequencing run could produce up to a terabase (Tb) of data. Using this technology, researchers are able to sequence up to five human genomes in a single run over a period of about one week. The cost is only about $5,000 per genome. How does NGS accomplish this?

In our case, it begins with a tissue sample. The DNA is extract from the nuclei of the cells in the sample and it is fragmented into millions of shorter double-stranded pieces. The ends of these pieces are repaired and adenylated so that adapter oligos can be ligated to both ends. The fragments are then size selected, purified, and separated into single strands. NGS uses a clustering technique performed on a flow cell. The flow cell has a dense lawn of oligos on its surface that bind to the adapters that were added to the fragments. Single-stranded DNA fragments are introduced to the flow cell, where they hybridize with the oligos present and undergo bridge amplification. The reverse strands are cleaved and washed away.

Hundreds of millions of clusters are sequenced simultaneously. The sequences are se-

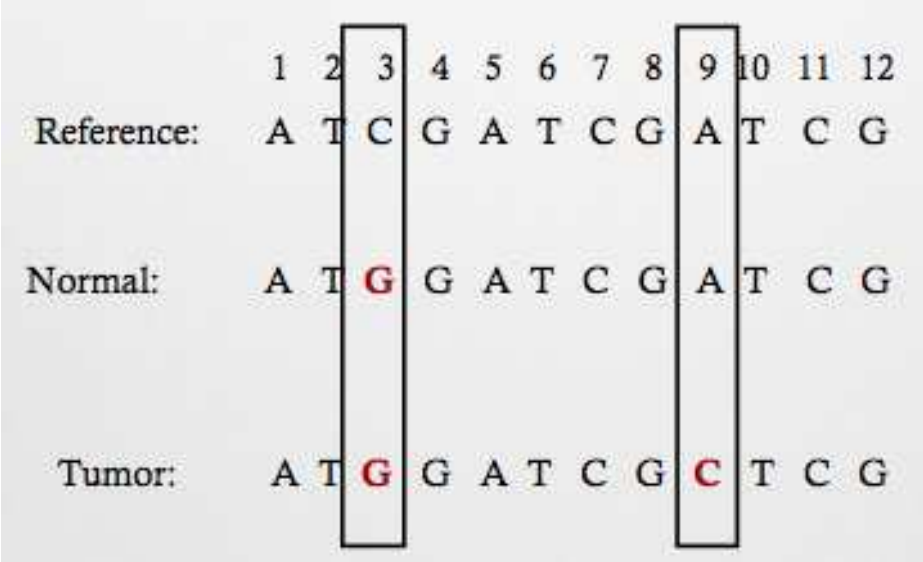


Figure 1.2: At position 3 we see a germline mutation. The mutation is present in both the normal tissue and the tumor tissue, indicating that the mutation was present at birth. At position 9 we see a somatic mutation. The normal tissue (Source: `http://www.stjude.org/stjude/v/index.jsp?vgnextoid=5817b5b36e70c210VgnVCM1000001e0215acRCRD`)

quenced base-by-base using fluorescently labeled, reversibly-terminated nucleotides. The four bases are each labeled with a different color for unique identification. These labeled nucleotides are washed over the surface of the flow cell, competing with one another to bind to the open positions within the clusters. After each cycle, the clusters are excited by a laser and emit the color of light corresponding to the base that was most recently added. The fluorescent label and blocking group are then removed, which allows the next base to be added.

1.2 Chromosome 1

The data given to us by BGI is a paired normal-tumor sequence of chromosome 1 from an individual with breast cancer. Chromosome 1 is the largest human chromosome, made up of about 249 million nucleotide base pairs. The tumor suppressor genes $BRCA1$ and $BRCA2$, located on chromosomes 17 and 13, respectively, have attracted particular attention as important factors in inherited forms of breast cancer. However, several oncogenes have been mapped to chromosome 1q as well, such as *NRAS*, *JUN*, *MYCL*, *TAL1*, and *BLYM*.

Chromosome 1 is associated with anomalies in 50% to 60% of breast tumors. In 1989, a karyotypic analysis of breast tumors revealed changes to chromosome 1q. Cytogenetic markers implicated a loss of heterozygosity in genes located on 1q23-32. (http://www.pnas.org/content/86/18/7204.full.pdf) Many breast cancer candidate genes for breast cancer have since been mapped to chromosome 1. For example: $NOTCH2$, $RHOC$, $GSTM1$, $RAP1A$, $MUC1$, $PARP1$. (http://www.cancerindex.org/geneweb/clinkc01.htm) A 2006 study by Orsetti et al. showed that the short arm shows frequent losses while the long arm shows frequent gains in breast tumors. (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2360604/)

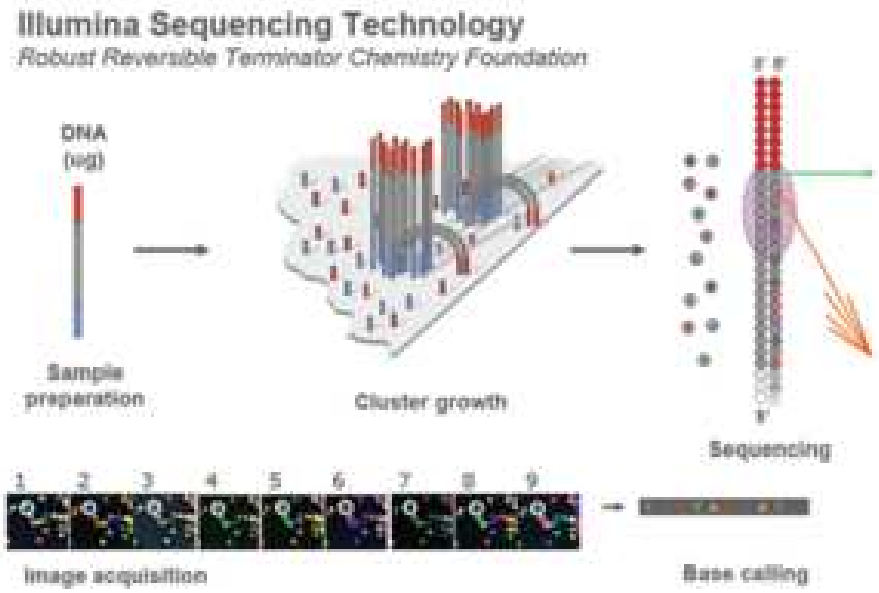


Figure 1.3: Next-generation sequencing uses three main steps: (1) library preparation, (2) clustering, and (3) sequencing.

With this in mind, it is important to correctly characterize the SNPs on chromosome 1 associated with breast cancer. Identification of SNPs is essential for candidate gene identification and further downstream analyses.

1.3 The Somatic Mutation Detection Pipeline

A pipeline is a linear sequence of specialized modules designed to accomplish a particular computing task. A typical bioinformatics pipeline is depicted in Fig. 1.4. First we sequence the normal and tumor DNA samples, producing short reads. Next we align these reads to a reference genome using tools such as SOAP2, BWT, or BOWTIE. The aligned sequences are compared using SAMtools or GATK to detect single nucleotide variants. These variants are then filtered by a somatic mutation detection program to filter out false positives and identify false negatives.

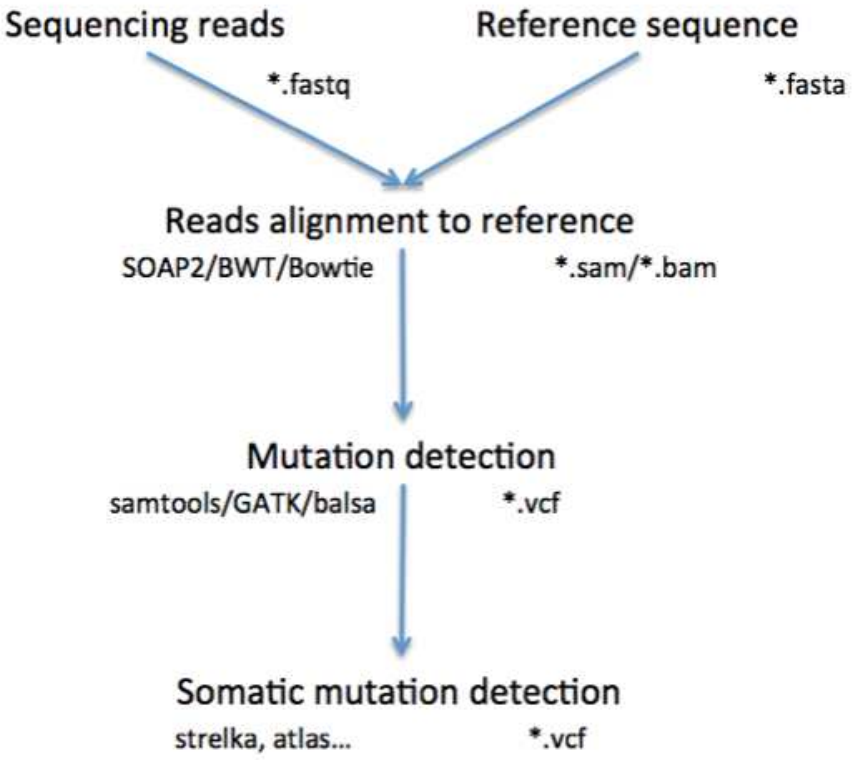


Figure 1.4: An example of a typical somatic mutation detection pipeline used by bioinformatics researchers. It begins with sequencing and aligning the paired normal/tumor samples to a reference genome. After the alignment, we identify mutation sites based on the sequenced data and then use some tool to determine which of these are truly somatic. (Figure courtesy of Binghang Liu.)

Chapter 2

The Task

2.1 Somatic Mutation Detection

Advances in high-throughput sequencing technologies allow for the detection of somatic mutations at a single nucleotide resolution. The process by which we detect true somatic mutations is referred to as "somatic mutation calling" and the tools we use for this purpose are "callers". Identification of these single nucleotide variant (SNV) somatic mutations is the first step in identifying the genes that drive cancer. Once these SNVs have been detected, we may conduct further downstream analyses, looking at how they alter gene expression or related pathways and understanding the clonal history of various tumor types. The main goal of this project is to accurately detect the somatic mutations present in the cancer genome.

However, the task is not as simple as comparing paired data from tumor and normal samples to a reference genome. There are sources of external error that impede our ability to declare with certainty that we have identified all of the somatic SNVs or that the SNVs we have identified are indeed somatic. We focus on four main sources of error: (1) sequencing errors, (2) sequencing bias, and (3) low mapping quality. In addition, no matter how good the average coverage of the sequencing run is, there may still be some portion of the genome with no reads at all. The coverage depth of a particular sequencing run is simply the average coverage depth over all the nucleotides, so regions with high coverage can average out with regions of very little coverage to no coverage at all. The basic challenge of somatic mutation calling involves navigating the uneven coverage of the sequencing operation.

2.1.1 Sequencing Errors

Sequencing errors occur when a sequencing method calls one or more bases incorrectly, which leads to an inaccurate read. Due to the vagaries of molecular biology, no DNA sequencing methods are perfectly accurate; some kinds of sequencing errors occur on the bases in reads in the machines.

Usually the chance of sequencing errors is estimable and quantifiable. Each base in a read is assigned a quality score, indicating the confidence of such base to be correctly. Some sequencing methods are more promised and reliable so they give higher quality scores for the bases. In addition, sequencing errors are also higher likely to happen in the end of a read, far from the insert has begun, so the quality scores will be lower.

Given variant pieces of reads of DNA sequences, we assign the reads in sequences following to the reference sequence. Therefore, on each locus, we have a column series of bases

that we denote it base pileups. In ideal cases, we expect at least 5 bases including in each base pileups in average. Assume on the certain locus, the reference has a base, say G, then we expect all the reads on this locus will be G. However, machines will not compute everything correctly, so it may return bases as A,C, or T instead of G. Such error is sequencing error. Sequencing errors are a major concern in detecting somatic mutation.

2.1.2 Sequencing Bias

DNA sequencing is subject to the flaws and limitations of the current sequencing technology. The data from BGI is produced is sequenced by Illumina next-generation sequencing technology. Illumina's sequencing technology is highly accurate, with excellent coverage and reasonable read lengths. However, it is also susceptible to particular forms of bias, for reasons known and unknown. It has been demonstrated that read coverage is correlated with GC composition in the sequence. That is, GC-rich and GC-poor regions show lower coverage, while more GC-balanced regions show higher coverage on average, leading to uneven coverage across the genome. The source of this technology's GC bias in unclear. What is clear is that uneven coverage causes problems for some genome assemblers, which tend to assume uniform coverage. [15]

Illumina next-generation sequencing also shows strand bias. Strand bias occurs when the genotypes inferred from the forward and reverse strands disagree. As an example, consider a fixed position in the genome. It is possible that the reads mapped to the forward strand support a heterozygous genotype while the reads mapped to the reverse strand support a homozygous genotype. This bias is caused by greater coverage on either the forward strand or the reverse strand, hence the term strand bias. As with GC bias, it is unclear what aspect of next-generation sequencing causes strand bias. Extreme strand bias has been shown to indicate the potential for high false-positive rates among somatic SNV callers. [14]

In addition, some sequences appear to be problematic for Illumina sequencers. The two main sources of sequence-specific error are inverted repeats and GGC sequences. These sequences interfere with the base elongation procedure used in next-generation sequencing. This phenomenon is a major source of coverage variability, a potential cause of false-positive calls, and a hinderance to assembly. [6] A successful somatic mutation caller must take these biases into account in identifying true somatic mutations.

2.1.3 Low Mapping Quality

Mapping quality scores quantify the probability that a read is misplaced. Alternatively, they may be interpreted as the probability that we align certain reads to the wrong loci. They are reported on the Phred scale.

Note that most of the alignments have a small likelihood of being mismatched, so this quantity will be effectively zero.

Here, on the mismatched bases, we consider the best possible genomes in the alignment. MAQ assigns each individual alignment a mapping quality. The mapping quality Q_s is the Phred probability that a read alignment may be wrong:

$$Q_s = -10 \log_{10} P(\text{read mapped incorrectly})$$

Therefore, $Q_s = 20$ means that the likelihood a read is aligned incorrectly is 1 in 100.

Here we have a reference sequence called X and a read sequence called Y. Assuming the sequences are independent at different sites of the read, the probability $P(z|X, Y)$, with sequence Y starting at position z is the product of error probabilities of the mismatched

bases at the alignment position. For example, if at position z, there are two mismatches with Phred quality 20 and 10, respectively, then $P(z|X,Y) = 10^{-(\frac{20+10}{10})} = 0.001$

To calculate the posterior probability $P_s(z|X,Y)$, we assume the uniform distribution $P(z|X)$, and apply Bayesian rules giving us

$$P_s(z|X,Y) = \frac{P(Y|X,z)}{\sum_{v=1}^{L-l+1} P(Y|X,v)}$$

where L is the length of X, $|X|$, and l is $|Y|$. Scaling P_s in the Phred way, we get the mapping quality of the alignment:

$$Q_s(z|X,Y) = -10\log_{10}[1 - P_s(z|X,Y)]$$

Chapter 3

Approach

3.1 Our Somatic Mutation Detection Pipeline

BGI has assigned us with the task of detection somatic mutations associated with a case of breast cancer. Hence our contribution to the typical pipeline can be thought of as an expansion of the "somatic mutation detection" module. We designed a somatic mutation detection pipeline to accomplish this task, shown in Fig. 3.1. We process the normal and tumor genomes separately in the beginning. First, we sort the BAM file in SAMtools and in GATK. Next, we index the sorted BAM file in SAMtools and GATK. Note that while we describe these processes in two steps, it actually only takes one step using GATK. Once the BAM file has been sorted and indexed, we identify putative somatic mutations using SAMtools' mpileup command and GATK's UnifiedGenotyper. After this step, we have four VCF files: two from the normal genome and two from the tumor genome. We use the VCF files to extract the features used in [**?**] and then merge the four sets of features to get the complete set.

All of this processing leaves us with the potential somatic mutations present in the paired normal/tumor sample with a set of 74 associated features for each variant site. From there, we run this data through a neural network to attempt to classify the mutations as somatic or non-somatic based on labeled training data from the Supplementary Materials of [5]. We also use principle component analysis (PCA) to reduce the dimension of our data and then apply various clustering techniques to separate the somatic mutations from the non-somatic ones.

3.2 From BAM to VCF

As mentioned previously, chromosome 1 is the largest human genome. As such, the normal and tumor BAM files are very large and too large to be processed all at once using the maximum memory available to the Ubuntu VMware player. Chromosome 1 is about 300 million base pairs in length, so we separated the reads in the original BAM files into twenty smaller BAM files based on region, each covering about 12 million base pairs. We can take only the reads overlapping a given region and direct them to another BAM file by running the following command in SAMtools:

```
samtools view -h <in.bam> <region>  >  <out.i.bam>
```

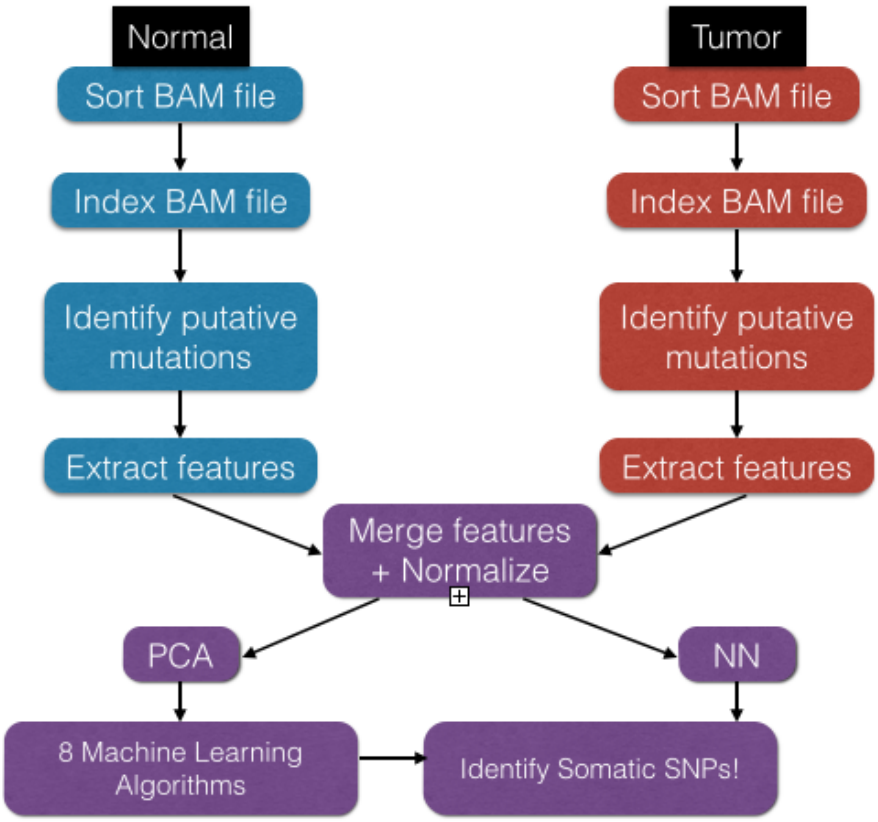


Figure 3.1: Our somatic mutation pipeline processes the normal and tumor genomes separately, first sorting and indexing them and then extracting features from the VCF files. Once we have the features, we merge the normal and tumor files and use classification techniques to try to identify the true somatic mutations.

where the region is formatted as

```
chr1:start-end
```

. The '-h' option outputs the header so that the output is a complete BAM file. However, in order to use the region option, the BAM file must be sorted and indexed. This means we must sort and index the entire BAM files before dividing them. Luckily, these are procedures are not as time or space intensive as the later steps. We sort and index the BAM files using the following SAMtools commands:

```
samtools sort <in.bam> <out.sorted.bam>
```

```
samtools index <in.sorted.bam>
```

Once we have sorted, indexed, and partitioned the file, we select one of the twenty smaller BAM files and run

```
mpileup
```

to create a BCF file. Since it is our job to detect the somatic mutations in a new way, we do not output only variants because that would be tantamount to using SAMtools as somatic mutation detection software.

```
samtools mpileup -gDSf <in.ref.fa> -Q 30 -r <region> <in.sorted.bam> > <out.bcf>
```

The output is in BCF format, so we have to pipe it into

```
bcftools
```

to get it into VCF format, which is human-readable in a text editor or Excel. We only consider bases with a quality score of greater than 30.

```
bcftools view <in.bcf> <out.vcf>
```

It is important that we do not use any options with 'view' command because otherwise it does not generate the I16 tag in the INFO section of the VCF file. We need the sixteen pieces of info contained in I16 for our feature construction.

Now we do a similar process using GATK. Take one of the smaller BAM files and sort and index it in the manner necessary fro GATK's UnifiedGenotyper using Picard tools:

```
java -jar path/to/AddOrReplaceReadGroups.jar \
> VALIDATION_STRINGENCY=SILENT \
> SORT_ORDER=coordinate \
> I=<in.bam> \
> O=<out.sorted.bam> \
> CREATE_INDEX=true
```

Once the file has been sorted and indexed, we may use GATK's UnifiedGenotyper to output the info needed for feature construction:

```
java -jar path/to/GenomeAnalysisTK.jar \
> -R <in.ref.fa> \
> -I <in.sorted.bam> \
> -o <out.vcf> \
> --output_mode EMIT_ALL_SITES \
> -stand_call_conf 30.0 \
> -nda \
> -slod \
> -L:capture,BED <interval.list>
```

Notice again that we choose to emit all sites and not just the variants. We do not consider bases with a quality score of less than 30. The interval list should be tab-delimited and of the form:

```
chr1 start end + interval_name
```

3.3 Features for Machine Learning

We used 62 features as input for our machine learning methods. Note that some come from SAMtools and some come from GATK. Some come from the normal sample and some come from the tumor sample. These components are brought together to make up the features for each SNP position. Most of the SAMtools features are taken from the I16 tag of the VCF file.

The final features we used were:

1. Number of reads covering or bridging the site in normal cell
2. Number of reference Q13 bases on the forward strand in normal cell
3. Number of reference Q13 bases on the reverse strand in normal cell
4. Sum of reference base qualities in normal cell
5. Sum of squares of reference base qualities in normal cell
6. Sum of reference mapping qualities in normal cell
7. Sum of squares of reference mapping qualities in normal cell
8. Sum of tail distances for reference bias in normal cell
9. Sum of squares of tail distances for non-reference bases
10. $P(D|G_i = aa)$, phred-scaled for normal cell
11. $\max_{G_i \neq aa} (P(D|G_i))$, phred-scaled in normal cell
12. $\sum_{G_i \neq aa} (P(D|G_i))$, phred-scaled in normal cell

13. Number of reads covering or bridging the site in tumor cell
14. Number of reference Q13 bases on the forward strand in tumor cell
15. Number of reference Q13 bases on the reverse strand in tumor cell
16. Number of non-reference Q13 bases on the forward strand in tumor cell
17. Number of non-reference Q13 bases on the reverse strand in tumor cell
18. Sum of reference base qualities in tumor cell
19. Sum of squares of reference base qualities in tumor cell
20. Sum of squares of non-reference base qualities
21. Sum of reference mapping qualities in tumor cell
22. Sum of squares of reference mapping qualities in tumor cell
23. Sum of non-reference mapping qualities in tumor cell
24. Sum of squares of non-refernce mapping qualities in tumor cell

25. Sum of tail distances for reference bases in tumor cell
26. Sum of squares of tail distance for reference bases in tumor cell
27. Sum of tail distances for non-reference bases in tumor cell
28. Sum of squares of tail distance for non-reference bases
29. $P(D|G_i = aa)$, phred-scaled, in tumor cell
30. $\max_{G_i \neq aa}\ (P(D|G_i))$, phred-scaled in tumor cell
31. $\sum_{G_i \neq aa}\ (P(D|G_i))$, phred-scaled in tumor cell

32. QUAL: phred-scaled probability of the call given data for normal cell
33. Total number of allueles in called genotypes in normal cell
34. Total (unfiltered) depth over all samples in normal cell
35. Fraction of reads containing spanning deletions in normal cell
36. HaplotypeScore: estimate the probability that the reads at this locus are coming from no more than 2 local haplotypes
37. MQ: root mean square mapping quality in normal cell
38. MQ0: total number of reads with mapping quality zero in normal cell in normal cell
39. QD: variant confidence/unfiltered depth in normal cell
40. SB: strand bias (the variation being seen only the forward or only the reverse strand) in normal cell
41. GQ: genotype quality computed based on the genotype likelihood in normal cell
42. $P(D|G_i = aa)$, phred-scaled in normal cell
43. $P(D|G_i = ab)$, phred-scaled in normal cell
44. $P(D|G_i = bb)$, phred-scaled in normal cell

45. QUAL: phred-scaled probability of the call given data for tumor cell
46. Allele count for non-ref allele in genotypes in tumor cell
47. AF: allele frequency for each non-ref allele in tumor cell
48. Total number of alleles in called genotypes
49. Total (unfiltered) depth over all samples
50. Fraction of reads containing spanning deletions in normal cell
51. HRun: largest contiguous homopolymer run of variant allele in either direction in tumor cell

52. HaplotypeScore: estimate the probability that the reads at this locus are coming from no more than 2 local haplotypes

53. MQ: root mean square mapping quality in normal cell

54. MQ0: total number of reads with mapping quality zero in normal cell in normal cell

55. QD: variant confidence/unfiltered depth in normal cell

56. SB: strand bias (the variation being seen only the forward or only the reverse strand) in normal cell

57. SumGLbyD in tumor cell

58. Allelic depths for the ref-allele

59. Allelic depths for the non-ref allele

60. DP: read depth (only filtered reads used for calling) in tumor cell item GQ: genotype quality computed based on the genotype likelihood in tumor cell

61. $P(D|G_i = aa)$, phred-scaled in tumor cell

62. $P(D|G_i = ab)$, phred-scaled in tumor cell

63. $P(D|G_i = bb)$, phred-scaled in tumor cell

3.4 Alignment and Merging

Once we have text files for both normal and tumor data, each containing features from SAMtools and GATK for each sorted 5% piece of positions, we read each position from both normal and tumor files, and eliminate those positions that are not SNPs. For those positions that are SNPs, we keep them and merge them together based on the aligned positions. Figure 3.2 here shows us the patterns of the merged file for aligned SNP positions. After this, for each aligned SNP position, we have the features from both normal and tumor parts while each part is composed by the SAMtools and GATK. Recall that there are 62 remaining features for each SNP which are constructed from SAMtools run on the normal data, SAMtools run on the tumor data, GATK run on normal data, and GATK run on tumor data. Once we have the aligned SNPs with all features for each 5% piece, we then merge all 20 pieces together into a whole text file including all SNPs detected from SAMtools and GATK. Note that we now have about 230,000 SNPs positions detected in total from 295 million total positions from the original data.

3.5 Missing Data and Mean Imputation

Notice that some of the SNP positions have too much missing data in features. Those SNP positions will affect the precision of our machine learning algorithms. Here Figure 3.3 is a screenshot for the values in different features and missing data (shown as dots). Our idea is to set up a cutoff value of the missing data for each SNP position so that if a certain SNP position has more than cutoff values of missing data in 62 features, we filter such SNP

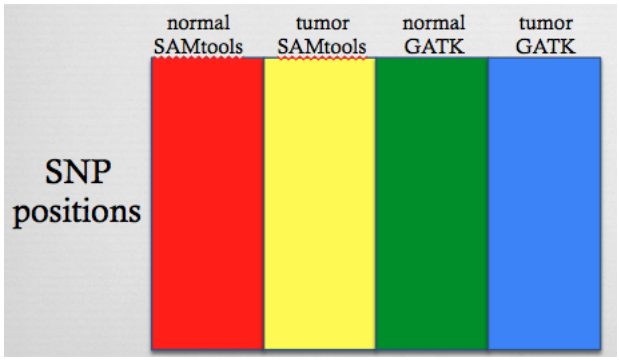


Figure 3.2: The manner by which we merged the features from each component into one set of features for each SNP.

positions. After this filtering procedure, we have 210,000 SNP positions left. While some of SNP positions still have several features with missing data, we do the mean imputation, which is to fill out the missing data part with the mean of that certain feature.

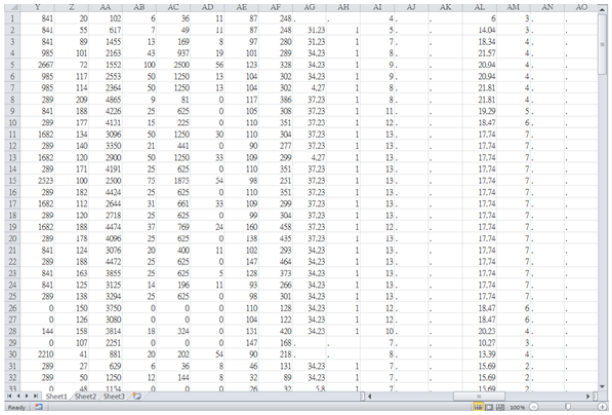


Figure 3.3: A sample of the missing data that we replaced by mean imputation. The '.' symbol represents a missing piece of data.

3.6 Detection of Experimental and Machine Errors

Since BGI′ data of VCF form has machine and experimental errors involved, such as the obvious over-bound values of Mapping Quality detected in some SNP positions, our next step is to filter those SNP positions with errors. Our method of detecting the experimental and machine errors is to compute the mean and standard deviation of each feature, and then for each certain feature value in each SNP position, we normalize it by substracting the mean and then dividing by the standard deviation. If a normalized value is larger than a cutoff value, we consider it as the experimental error or machine error because is

so far removed from the others. Once a SNP position has features that have the labels of experimental and machine errors, we filter that SNP position. This procedure reduces the size of our data to nearly 180,000 SNP positions.

3.7 Normalization

In order to guarantee the precision of the data training procedure in machine learning algorithms, our input data to machine learning algorithms are normalized. Our method of normalization is to compute the maximum value of the absolute values of terms for each feature, and then divide each value of that feature in each specific SNP position by that maximum absolute value for that feature so that the output value hits in the range of -1 to 1.

3.8 Principle Component Analysis (PCA)

PCA is method of dimensionality reduction which uses an orthogonal transformation to produce a orthonormal basis for the given data. These new linearly uncorrelated variables are referred to as principle components and they have the largest possibility variance given the constraint of orthonormality. That is, the first principle component accounts for as much of the data's variability as possible and then the second principle component accounts for as much of the variability as possible while remaining orthonormal to the first principle component, and so on. We may think of PCA as providing a lower-dimensional projection of high-dimensional data when viewed from its most informative angle.

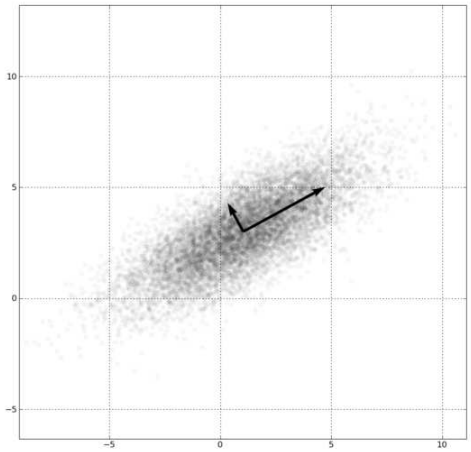


Figure 3.4: Using PCA, the eigenvectors of the covariance matrix are computed and scaled by the square root of the corresponding eigenvalue. The first two principle components are shown as vectors in the figure. Notice how they are in the directions of greatest variance while remaining orthogonal to each other. (Source: `wikipedia.org/wiki/Principal_component_analysis`)

We give an intuitive description of the steps involved in PCA. Suppose we have a data

matrix X with dimensions $m \times p$ where m is the number of observations and p is the number of features. We take the values in each column of this matrix to be mean-centered. We calculate the covariance matrix as $XX^\intercal$. The covariance matrix is symmetric and hence diagonalizable, so we can write

$$XX^\intercal = WDW^\intercal$$

. However, in practice, we often use singular value decomposition (SVD) of X to get $U\Sigma V^\intercal$. Once we have this formulation, computing the covariance matrix is simple:

$$XX^\intercal = (U\Sigma V^\intercal)(V\Sigma U^\intercal) = U\Sigma^2 U^\intercal,$$

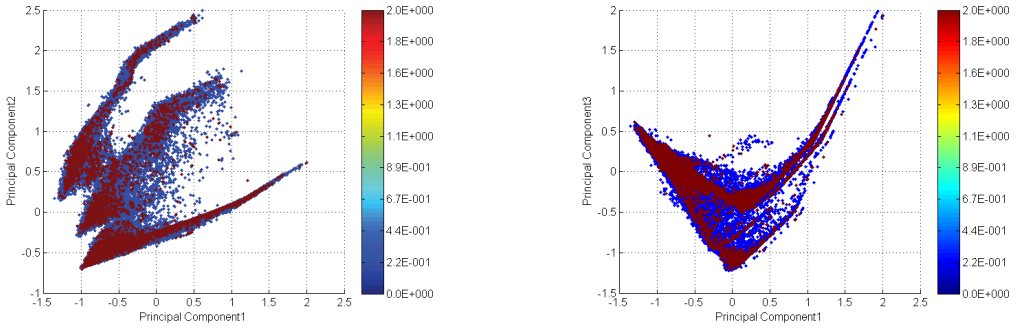
since V is an orthogonal matrix. This is useful for matrices which are amenable to SVD but not to direct calculation of $XX^\intercal$. Once we have the covariance matrix, we calculate the eigenvalues and corresponding eigenvectors. Now we sort the eigenvectors by decreasing eigenvalue and select the first k, where k is the desired dimension after dimensionality reduction. Forming an eigenvector matrix from the k selected eigenvectors, we transform the higher-dimensional data into k-dimensional data through matrix multiplication.

3.9 PCA on the Pre-labeled Data

We are interested in finding the principal components' effect on the variance in our data with the pre-labels determined by SAMtools and GATK. For the 62 features for each SNP, we perform PCA and then choose the top 10 principal components of the features' linear combination. Here we choose the weight of the principal components for plotting. We pre-label each SNP position into one of the two colors: blue means either SAMtools or GATK returns a result that such feature is somatic mutation, while the red color is a result for germline mutation shown from SAMtools and GATK. Notice that this result is not the accurate somatic or non-somatic result for SNPs since the results from SAMtools and GATK are also with sequencing errors or sequencing bias. Here we plot all the SNP positions based on the weight of the two principal components. In Figure 3.5, we show the Principal Component 1 against Principal Component 2, PC1 vs. PC3, and PC2 vs. PC4, respectively. As shown, the cluster between somatic and non-somatic mutation parts based on the detection of SAMtools and GATK are not clear enough. In the perfect world, we expect to see a clear cluster can classify all the SNPs into two parts. In fact, our machine learning algorithms show us the clear results and we will state in the later section.

3.10 Classification Using Nine Machine Learning Algorithms

Once we have used PCA on our data, we are left with a list of principle components, ordered by the amount of variance they capture. The first ten principle components captured a total of 81.5% of the variance in our data, so we used these ten components as features in SciKit, a machine learning package in Python. With SciKit, we could run nine machine learning algorithms at once and output the results from each. We also used an artificial neural network, with 62 input nodes representing the features described in Section 3.3. The output from each of these algorithms is a binary classifier, with 1 representing a somatic mutation and 0 representing a non-somatic mutation.



(a) PC 1 vs. PC 2

(b) PC 1 vs. PC 3

(c) PC 2 vs. PC 4

Figure 3.5: The patient's SNP data plotted using pairs of principle components that separate the data well visually. In our machine learning algorithm, we use the first ten principle components as features. Blue represents that the SNP was labeled as somatic by SAMtools, while red represents a SNP labeled as germline by SAMtools.

We trained the data using labeled training data from [5]. This can be obtained from `http://bioinformatics.oxfordjournals.org/content/28/2/167.long`. Go to Supplementary Data and download the Supplementary Data zip file.

Our Scikit and neural network output combined gives each SNP a list of nine bits, representing whether each algorithm assigned it a label of somatic or non-somatic, using an activation threshold of 0.9. We assigned those SNPs which at least 8 algorithms assigned a label of somatic and final label of somatic. All others were labeled non-somatic. After this procedure, we were left with about 56,000 somatic SNPs. We extract the positions of these SNPs into a text file using get_positions.m (code found in Appendix A). The results of the classification for each method are shown in Fig. 3.6.

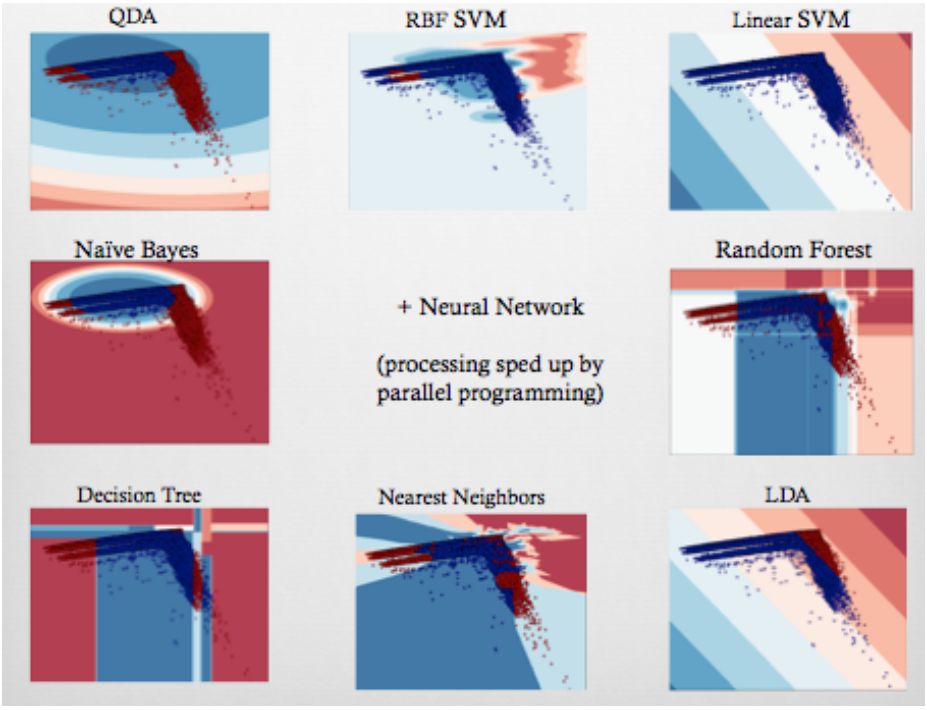


Figure 3.6: A visualization of the classification of our SNPs using the 8 algorithms in the SciKit package. Note that we visualized this using only 2 principle components, but in actuality we classified them using 10 principle components. Neural network not shown.

Now we describe the algorithms used in a bit more detail. More resources on these algorithms can be found in [13].

3.10.1 Description of Algorithms

1. **Support Vector Machine (SVM)**: SVMs are maximum margin methods that allow the model to be written as a sum of the influences of a subset of the training instances. These influences are given by application-specific similarity kernels. See Fig. 3.7. We use two versions of SVM: one with a linear kernel and one with a radial basis function (RBF) kernel.

2. **Linear Discriminant Analysis (LDA)**: LDA is a supervised learning method for dimensionality reduction for classification problems. We start with the case where

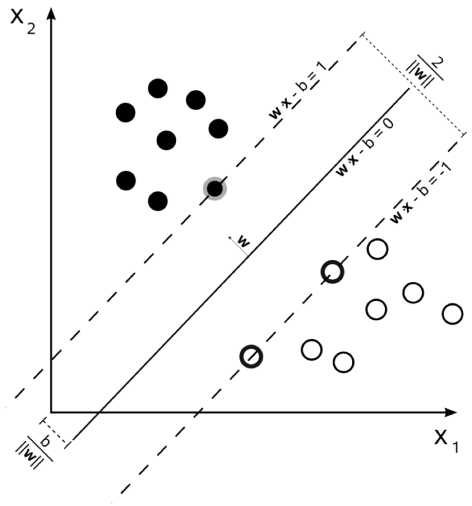


Figure 3.7: An example of classification using a support vector machine (SVM) with a linear kernel.

there are two classes, then generalize to K ¿ 2 classes. Given samples from two classes $C1$ and $C2$, we want to find the direction, as defined by a vector w, such that when the data are projected onto w, the examples from the two classes are as well separated as possible.

3. Quadratic discriminant analysis (QDA): Closely related to LDA, QDA does not assume that the covariance between classes is identical. The surfaces separating the classes will be conic sections as opposed to a linear model.

4. **K-Nearest Neighbors (KNN)**: The training samples and the testing samples are vectors in multidimensional feature space. The training samples have their class label. Every testing sample is classified by assigning the label which is most frequent among the K nearest training samples. Euclidean distance is commonly used for continuous variables. Other distance metric may applied to the same data set to gain different label thus form a different classification. PCA can serve as a pre-processing step for K-nearest neighbors algorithm with feature selection and dimension reduction combined in one step.

5. **Decision Tree**: A decision tree is a decision-making tool using tree-like model to predict the consequence or class of the testing samples. A decision tree is a simple representation for classifying examples. Decision tree learning is one of the most successful techniques for supervised classification learning A decision tree has 2 kinds of nodes: (1) Each leaf node has a class label, determined by majority vote of training examples reaching that leaf. Each leaf of the tree is labeled with a class or a probability distribution over the classes. (2) Each internal node is a question on features. It branches out according to the answers. After training the decision tree model, the label will be assigned to each testing sample by the going through the tree based on the features of the testing samples.

6. **Random forest**: Random forests are ensemble learning method for classification. Several decision trees are trained at the training time. The training algorithm for

random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Each time, take n training examples from X, Y with replacement and these subsets of training set X_i, Y_i are used to train a decision tree, thus having several decision trees with different training data. After training, predictions for unseen samples can be made by averaging the predictions from all the individual regression trees on or by taking the majority vote in the case of decision trees

7. **Naive Bayes**: Naive bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes is a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate preprocessing, it is competitive in this domain with more advanced methods including support vector machines.

8. **Neural Network (NN)**: The multilayer perceptron is an artificial neural network structure and is a nonparametric estimator that can be used for classification and regression. The back-propagation algorithm is used to train a multilayer perceptron for a variety of applications. See Fig. 3.8.

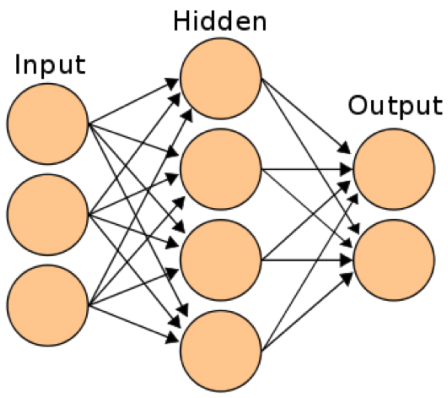


Figure 3.8: An example of a multilayer perceptron with 3 input nodes (features), 4 hidden nodes, and 2 output nodes.

3.11 Speeding Up the Neural Network

3.11.1 Parallel computing with OpenMP

OpenMP provides a straight-forward interface to write software that can use multiple cores of a computer. Using OpenMP you can write code that uses all of the cores in a multicore computer, and that will run faster as more cores become available. OpenMP is a well-established, standard method of writing parallel programs. It was first released in 1997, and is currently on version 3.0. It is provided by default in nearly all compilers, e.g. the gnu compiler suite (gcc, g++, gfortran), the Intel compilers (icc, icpc, ifort) and the Portland Group compilers (pgc, pgCC, pgf77) and works on nearly all operating systems (e.g. Linux, Windows and OS X).

One nice property of Neural Network is that to update the value of each node within a certain layer is completely independent of each other node within that layer. Hence we can distribute this task to different threads using OpenMP.

Chapter 4

Conclusions

4.1 SNP Annotation

Following our classification by nine machine learning algorithms, we have around 56,000 SNPs that we identified as somatic. Among these, there are SNPs that were originally classified as non-somatic (false negatives) and SNPs that were originally classified as somatic (true positives). The distribution of these classes is approximately 46,000 and 10,000, respectively. We choose to assess our results through the use of a SNP annotation tool called SNPnexus. Given a list of SNP positions with corresponding reference and alternate alleles, this tool compiles functional information from a host of genetic databases. SNPnexus can provide genomic coordinates, gene and protein consequences, effect on protein function, HapMap population data, and (most importantly for us) phenotype and disease associations. Unfortunately, the true genotype of our patient is not available to us, nor could it be. We will always to subject to the uncertainty inherent to sequencing technology. With that in mind, we cannot simply check that SNPs we found are correct. Indeed, if we could do that, there would have been no impetus for our work in the first place. Instead we use SNP annotation as a way to reference the SNPs we found against those found by other researchers. One way to measure our success is to see how many of our somatic SNPs are assigned a dbSNP ID. However, this only serves to show that a particular SNP has been seen before by other researcher. Given the randomness of somatic mutations and the enormous number of possibilities, this number is unlikely to carry much meaning. A more salient point would be made if we could show that a subset of the SNPs we identified were found in other breast cancer patients or if many SNPs were found in breast cancer driver genes.

SNPnexus accepts a VCF file as input and we have VCF files for the entirety of chromosome 1, so that seemed to be the simplest method. We wrote a Java program to trim down the VCF file to contain only the positions we identified as somatic mutations. See Appendix A for VcfTrimmer.java. This code reads in the VCF file one line at a time, gets the position token and checks if that position is in our list of somatic SNPs. If it is, we keep the whole line. Otherwise, we throw it out. After running this code, we are left with a VCF file containing only the positions we are concerned with and it is ready to be uploaded to SNPnexus.

We ran SNPnexus with the following options:

- Assembly: GRCh37/hg19
- Query Type: Batch Query

- Batch Query: (uploaded VCF file)
- Gene/Protein Consequences: RefSeq, Ensembl, UCSC
- Phenotype & Disease Association: Genetic Association of Complex Diseases and Disorders (GAD), Catalogue of Somatic Mutations in Cancer (COSMIC), NHGRI Catalogue of Published Genome-Wide Association Studies
- Zipped content: Text

We waited about twelve hours for the results and then began our analysis. Each of the databases returns a table which can be saved as a text file. We wrote a Java program (see Appendix A for AnnotationParser.java) that takes these tables and performs the following set of analyses:

1. Gets the dbSNP identification rate. Of course, our method identifies some "germline" or "wildtype" mutations as somatic. These could be false negatives due to sequencing errors. However, we won't count these in calculating our match rate because they can't be checked in SNP database at all.
2. Checks the tables from the gene consequences databases for known breast cancer driver genes on chromosome 1. We compiled a large list from several PubMed papers, found in Appendix B. (3) Searches the tables from the phenotype and disease association databases for keywords such as "breast" and "cancer".

The database match rate was .197. We found 8,660 associations with breast cancer in other studies (note that this does not mean that 8,660 different SNPs were associated with breast cancer. Rather, one SNP might be cited in several studies). We also found 398 mutations in three genes known to be breast cancer driver genes: PARP1 [234 SNPs], MAP1A [151 SNPs], and TACSTD2 [13 SNPs].

4.2 Discussion

Of the approximately 10,000 SNPs we classified as true positives, 19.7% of them were assigned a dbSNP ID. If we think of this as a "classification rate" (which we should not), then this is very poor percentage. Far from being a classification rate, this number is best interpreted as the percentage of the SNPs we identified that have been found in enough other research studies that they have been assigned an identification number. Considered in this way, the number is far more reasonable. There are 295 million positions on chromosome 1. Each position has a reference allele assigned to it, so that leaves three choices of alternate nucleotides. Hence there are about 885 million possible SNPs on chromosome 1 and not every possibility is present in the dbSNP database. Therefore our the SNPs we identified in our patient may truly be present, but they were not in the dbSNP database.

The 8,660 associations serves as an indicator that we successfully identified breast cancer driver SNPs because other breast cancer patients had these SNPs as well. If we had more time, we would look at the number of distinct SNPs covered by these 8,660 associations and see what genes they were present in.

We now examine the cancer driver genes in which we found mutations in more detail, beginning with PARP1. The PARP1 gene codes for an enzyme called poly[ADP-ribose] polymerase 1. PARP1, as its name suggests, modifies nuclear proteins through

ADP-ribosylation (see Fig. 4.1). It works in conjunction with BRCA1 (breast cancer 2, early onset) and BRCA2 (breast cancer 2, early onset), two human tumor suppressor genes, and plays a role in DNA damage repair as well. The enzyme repairs single-stranded DNA (ssDNA) breaks. Inactivation of PARP1, whether by mutations or siRNA knockdowns, impairs the cells' ability to repair ssDNA breaks and so during DNA replication the process comes to a halt at these breaks. If the homologous recombination (HR) pathway is still functioning, then the breaks will be repaired that way instead. If, however, the HR pathway fails, then ssDNA breaks accumulate and we observe an increased incidence of tumor formation. It would be interesting to check whether this patient has any mutations in genes involved in HR, resulting in a non-functional HR pathway. The interaction with BRCA1 and BRCA2 is significant because the BRCA enzymes act on repair of double-stranded DNA breaks and so breast cancers with BRCAness (tumor has mutations in either BRCA1 or BRCA2) are highly sensitive to inhibition of PARP1. A suggested therapy for cancer with BRCAness proposes PARP1 inhibitors, which would cause the tumor cell to undergo programmed cell death, called apoptosis, due to the amount of DNA damage.

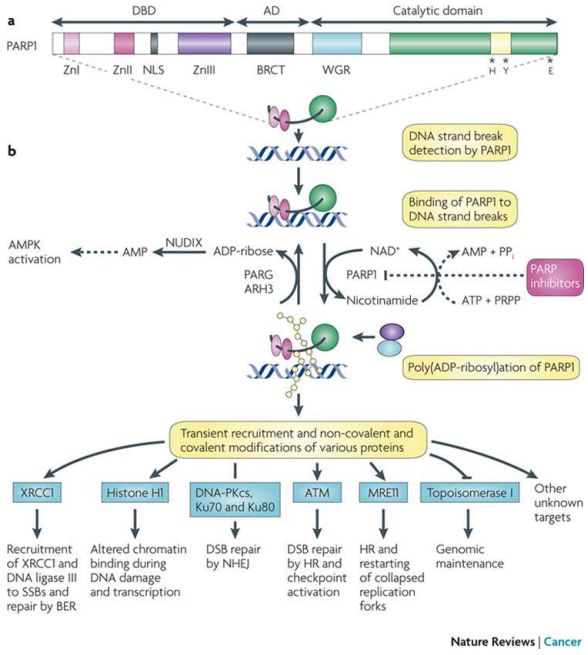


Figure 4.1: PARP1 starts a signaling cascade involved in the repair of ssDNA breaks. (Source: `http://www.nature.com/nrc/journal/v10/n4/fig_tab/nrc2812_F1.html`)

MAP1A codes for microtubule-associated protein 1A. Proteins in this family are involved in microtubule assembly and are distributed across the cell (Fig. 4.2). Microtubules are components of the cellular cytoskeleton and are involved in many cellular processes. They maintain the structure of the cell, provide a method of intracellular transport, and are essential to cell division. One class of cancer-fighting drugs, referred to as taxanes, block dynamic instability by stabilizing GDP-bound tubulin. This prevents GDP from being phosphorylated into GTP, which would cause the microtubule to shrink. However, this protein is expressed primarily in the brain and is implicated in neurogenesis. Its association with breast cancer may be indirect, associative, or perhaps just not fully understood.

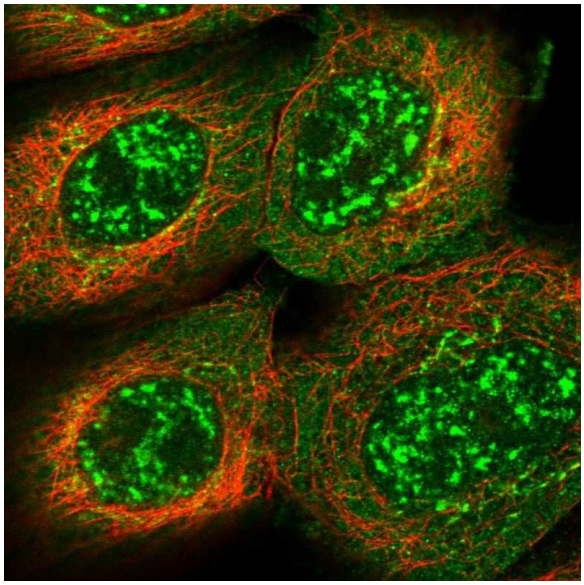


Figure 4.2: (The distribution MAP1A across the cell, visualized using fluorescent labels. Note the presence on the microtubules. Source: `http://www.novusbio.com/MAP1A-Antibody_NBP1-82801.html`)

TACSTD2 encodes the tumor-associated calcium signal transducer 2 protein. The protein is an cell-surface receptor antigen (involved in immune response) responsible for transducing an intracellular calcium signal. The antigen is carcinoma-associated, but the mechanisms by which mutations in this gene give rise to tumors in epithelial cells. Their distribution across the cell surface is shown in Fig. 4.3.

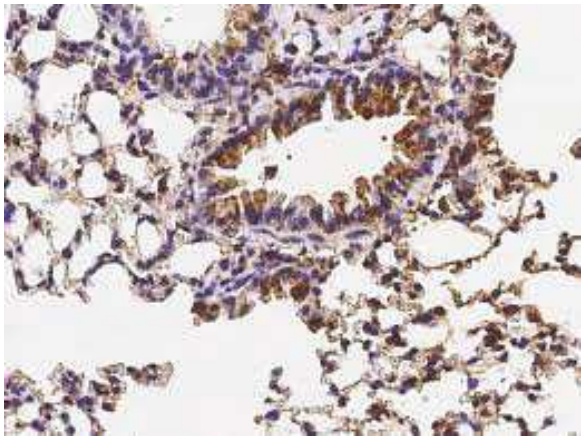


Figure 4.3: TACSTD2 present on the cell surface as a receptor. (Source: `http://old.sinobiological.com/TROP-2-TACSTD2-Antibody-g-17655.html`)

4.3 Suggestions for Future Work

We ran into an issue of overfitting, a term used to describe when a model describes noise instead of the underlying relationship, because our training data ($\sim 1,000$ SNPs) was very small relative to our testing set ($\sim 180,000$ SNPs). If we could compile a large repository of ultra-deep sequenced exon (and possibly intron) data from chromosome 1, then upon training the machine learning algorithms we would better capture the intrinsic qualities of somatic and non-somatic SNPs rather than fitting to one specific patient.

Appendix A

Project Code

```
%% This program takes the output of SciKit, as modified for
%% the purposes of RIPS-BGI, and selects only the SNP positions
%% which more than some threshold number of algorithms classified
%% as somatic. For example, if the input parameter is 4, then we
%% return a list of positions that 4 or more algorithms classified
%% as the site of a somatic mutation.

function get_positions(scikit_output, threshold)

%% Instance variables
SUM_COLUMN = 10;

%% Load the scikit output
A = csvread(scikit_output);

%% For each row, look at the sum. Return the row if it is more than
%% the threshold.
while i <= size(A,1)
    if A(i,10) < threshold
        A(i,:) = [];
    else
        i++
    end
end

%% Print the position column to a csv document
csvwrite('threshold_positions.csv', A(:,11));
end
```

```
function pca_plotter2
% load .csv file into a matrix
% A = load(file_name);
B = xlsread('Normalization_Done_1.xlsx');
[m n] = size(B);
A=B(:,1:n-2);
% run PCA on all columns except the last (this is the label column)
[COEFF, SCORES, latent, tsquare] = princomp(A);
[m n] = size(A);
% How much of the variance is accounted for by the first two principle
% components? How about the first three?
%% latent: the dominant eigenvalues vector
variance = cumsum(latent)./sum(latent);
eigenval1 = variance(1)
eigenval2 = variance(2)
eigenval3 = variance(3)

%% print the result for the first two principal component
Vector = [SCORES(:,1),SCORES(:,2)];
csvwrite('PrinciComp_1.csv', Vector);

k=1;
%% plot data
figure
for i=1:4
    for j=(i+1):5
        %string =
            strcat('label_vector_1.',num2str(i),num2str(j),'.xlsx');
%
    plotclr(SCORES(1:m,i),SCORES(1:m,j),xlsread('label_VEctor.1.xlsx'));
        subplot(2,5,k);
        plotclr(SCORES(1:m,i),SCORES(1:m,j),xlsread('label_VEctor.1.xlsx'));
%         plotc(SCORES(1:m,i),SCORES(1:m,j),'.');
        string1 = strcat('Principal Component ',num2str(i));
        string2 = strcat('Principal Component ',num2str(j));
        xlabel(string1);
        ylabel(string2);
        k=k+1;
        %hold on
    end
end
```

```
%% program for merging normal and tumor files
pos_col = 1;                    %% POS column
alt_col_NS = 12;%% ALT column
alt_col_NG = 27;
alt_col_TS = 20;
alt_col_TG = 39;

NR_file = fopen('normal.real.1.csv');
TR_file = fopen('tumor.real.1.csv');

textLineN = fgetl(NR_file);
textLineT = fgetl(TR_file);

output_fileN = fopen('writeNormal.1.csv','w');
output_fileT = fopen('writeTumor.1.csv','w');

numLine=0;
tic;
while (ischar(textLineN)==1) && (ischar(textLineT)==1)
    tokenN = textscan(textLineN,'%s','delimiter',',');
    tokenN = tokenN{1,1};
    tokenN = transpose(tokenN);
    tokenT = textscan(textLineT,'%s', 'delimiter', ',');
    tokenT = tokenT{1,1};
    tokenT = transpose(tokenT);
    posN = str2double(tokenN(1));
    posT = str2double(tokenT(1));
     if posN == posT
               fprintf(output_fileT,'%s\n',textLineT);
               if (~feof(NR_file))&&(~feof(TR_file))
                   textLineN = fgetl(NR_file);
                   textLineT = fgetl(TR_file);
                   tokenN = cell(1,30);
                   tokenT = cell(1,42);
                   numLine=numLine+1;
               else break;
               end
     elseif posN > posT
         if ~feof(TR_file)
             textLineT = fgetl(TR_file);
             tokenT = cell(1,42);
         else break;
         end
     else %%posN<posT
         if ~feof(NR_file)
             textLineN = fgetl(NR_file);
             tokenN = cell(1,30);
         else break;
```

```
        end
    end
     %% get the next line and refresh
end

toc;

%% close the files
fclose(output_fileN);
fclose(output_fileT);

%% next procedure: Throw out positions that are not SNP sites in either
   the normal or the tumor files
nuMLine=0;
NR_file = fopen('writeNormal.1.csv');
TR_file = fopen('writeTumor.1.csv');
textLineN = fgetl(NR_file);
textLineT = fgetl(TR_file);
output_fileN = fopen('condensedNormal.1.csv','w');
output_fileT = fopen('condensedTumor.1.csv','w');

while (ischar(textLineN)==1) && (ischar(textLineT)==1)
    tokenN = textscan(textLineN,'%s','delimiter',',');
    tokenN = tokenN{1,1};
    tokenN = transpose(tokenN);
    tokenT = textscan(textLineT,'%s', 'delimiter', ',');
    tokenT = tokenT{1,1};
    tokenT = transpose(tokenT);
    altNS = str2double(tokenN(12));
    altNG = str2double(tokenN(27));
    altTS = str2double(tokenT(20));
    altTG = str2double(tokenT(39));
    if (altNS ~=0) || (altNG~=0) || (altTS~=0) || (altTG~=0)
        fprintf(output_fileT,'%s\n',textLineT);
        if ~feof(NR_file) %% &&(~feof(TR_file))
            textLineN = fgetl(NR_file);
            textLineT = fgetl(TR_file);
            tokenN = cell(1,30);
            tokenT = cell(1,42);
            nuMLine=nuMLine+1;
        else break;
        end
    else
        if ~feof(TR_file)
            textLineT = fgetl(TR_file);
            tokenT = cell(1,42);
            textLineN = fgetl(NR_file);
            tokenN = cell(1,30);
        else break;
```

```
        end
    end
end
fclose(output_fileN);
fclose(output_fileT);

A = xlsread('condensedNormal.1.csv');
B = xlsread('condensedTumor.1.csv');

%% Throw out positions that are not SNP sites in either the normal or
    the tumor files
i = 1;
while i <= size(A(:,1),1)
    if A(i,alt_col_NS)==0 && A(i, alt_col_NG)==0 && B(i,alt_col_TS)==0
        && B(i,alt_col_TG)==0
        A(i,:) = [];
        B(i,:) = [];
    else
        i = i + 1;
    end
end
%% print the size of normal and tumor matrices
[x y] = size(A)
[a b] = size(B)
%% record the vector D for assumed labels
D=zeros(x,1);
tic;
for i=1:x
    if  A(i,alt_col_NS)~=B(i,alt_col_TS) ||
        A(i,alt_col_NG)~=B(i,alt_col_TG)
        %%(A(i,alt_col_12)~=0 && B(i,alt_col_20)~=0) &&
            (A(i,alt_col_27)~=0 && B(i,alt_col_39)~=0)
              %% case for germline
        D(i)=0.2; %% somatic mutation is red
    else
        D(i)=2; %% germline is blue
    end
end
D(x) = 0;

xlswrite('label_VECtor.1.xlsx', D);
toc;

%% C: the matrix which includes all the features we need
C=zeros(x,64);

%% first 20 features for normal by SAMtools; next 20 features for tumor
    by GATK
C(:,1:9) = A(:,2:10);
```

```
C(:,10:12) = A(:,13:15);
C(:,13:29) = B(:, 2:18);
C(:,30:32) = B(:, 21:23);
%%
C(:,33:42) = A(:,16:25);
C(:,43:45) = A(:,28:30);
C(:,46:59) = B(:,24:37);
C(:,60:62) = B(:,40:42);
C(:,63) = round(rand(x,1));
C(:,64) = A(:,1);
%% sorting and finding method

%% print out C which lists all the features from SAMtools and GATK for
    both normal and tumor
xlswrite('real.emerged.1.xlsx',C);
```

```
%% super merging file for 1 to 19 pieces
tic;
A = xlsread('real.emerged.1.xlsx');
B = xlsread('real.emerged.2.xlsx');
C = xlsread('real.emerged.3.xlsx');
D = xlsread('real.emerged.4.xlsx');
E = xlsread('real.emerged.5.xlsx');
F = xlsread('real.emerged.6.xlsx');
G = xlsread('real.emerged.7.xlsx');
H = xlsread('real.emerged.8.xlsx');
I = xlsread('real.emerged.9.xlsx');
J = xlsread('real.emerged.10.xlsx');
K = xlsread('real.emerged.12.xlsx');
L = xlsread('real.emerged.13.xlsx');
M = xlsread('real.emerged.14.xlsx');
N = xlsread('real.emerged.15.xlsx');
O = xlsread('real.emerged.16.xlsx');
P = xlsread('real.emerged.17.xlsx');
Q = xlsread('real.emerged.18.xlsx');
R = xlsread('real.emerged.19.xlsx');
Z = [A;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R];
[m n]=size(Z)
toc;
xlswrite('real.emerged.xlsx', Z);
```

```
%% super merging file for 1 to 19 pieces
tic;
A = xlsread('label_VECtor.1.xlsx');
B = xlsread('label_VECtor.2.xlsx');
C = xlsread('label_VECtor.3.xlsx');
D = xlsread('label_VECtor.4.xlsx');
E = xlsread('label_VECtor.5.xlsx');
F = xlsread('label_VECtor.6.xlsx');
G = xlsread('label_VECtor.7.xlsx');
H = xlsread('label_VECtor.8.xlsx');
I = xlsread('label_VECtor.9.xlsx');
J = xlsread('label_VECtor.10.xlsx');
K = xlsread('label_VECtor.12.xlsx');
L = xlsread('label_VECtor.13.xlsx');
M = xlsread('label_VECtor.14.xlsx');
N = xlsread('label_VECtor.15.xlsx');
O = xlsread('label_VECtor.16.xlsx');
P = xlsread('label_VECtor.17.xlsx');
Q = xlsread('label_VECtor.18.xlsx');
R = xlsread('label_VECtor.19.xlsx');
Z = [A;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R];
[m n]=size(Z)
toc;
xlswrite('label.emerged.xlsx', Z);
```

```
%% read in the file
%% normalization with the imputation
function Normalization
w = input('Normalization method number: ')
A = xlsread('real.emerged.master.xlsx');
[m h] = size(A) %% m=808 n=65 807*64
n=h-2;
B = A(:,1:n);
%% imputation procedure for average
tic;
B(find(isnan(B)))=0;
AVG = mean(B);
%size(AVG)
for i=1:m
    for j=1:n
        if B(i,j)==0
            B(i,j) = AVG(j);
        end
    end
end
toc;
%%
if w==1
    %% normalization 1
    tic;
    C = B-repmat(mean(B),m,1);
    d = std(B);
    D = repmat(d,m,1);
    i=1;
    counter=1;
    t=n;
    while i <= t
        if d(counter)==0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C=C./D;
    toc;

elseif w == 2
    %% normalization 2
    tic;
    C=B;
```

```
    i=1;
    t=n;
    counter=1;
    d = max(abs(B));
    D = repmat(d,m,1);
    while i <= t
        if d(counter) == 0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C = C./D;
    toc;
end
%%result
C = [C,A(:,(h-1):h)];
xlswrite('Normalization_Imputation_screening_std.xlsx', C);
```

```
%% read in the file
%% normalization with the imputation
function Normalization_simulation
w = input('Normalization method number: ')
%B = xlsread('sam_emerged.xlsx');
A = xlsread('simulated_data.xlsx');
[m n] = size(A) %% m=808 n=65 807*64
B = A(:,1:n-1);
%% imputation procedure for average
tic;
AVG = mean(B);
%size(AVG)
for i=1:m
    for j=1:n-1
        if B(i,j)==0
            B(i,j) = AVG(j);
        end
    end
end
toc;
%%
if w==1
    %% normalization 1
    tic;
    C = B-repmat(mean(B),m,1);
    d = std(B);
    D = repmat(d,m,1);
    i=1;
    counter=1;
    t=n-1;
    while i <= t
        if d(counter)==0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C=C./D;
    toc;

elseif w == 2
    %% normalization 2
    tic;
    C=B;
    i=1;
```

```
    t=n-1;
    counter=1;
    d = max(abs(B));
    D = repmat(d,m,1);
    while i <= t
        if d(counter) == 0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C = C./D;
    toc;
end
%%result
%xlswrite('Normalization_Done.xlsx', C);
A=[C,A(:,n)];
xlswrite('Normalization_Simulation.xlsx', A);
```

```
%% read in the file
%% normalization with the imputation
function normalization_std_filtering
w = input('Normalization method number: ')
A = xlsread('real.emerged.xlsx');
X = xlsread('label.emerged.xlsx');
[m h] = size(A) %% m=808 n=65 807*64
[f g] = size(X)
%% m is the number of length for the labels
n=h-2;
B = A(:,1:n); %%n=62
%% imputation procedure for average
tic;
B(find(isnan(B)))=0;
AVG = mean(B);
%size(AVG)
for i=1:m
    for j=1:n
        if B(i,j)==0
            B(i,j) = AVG(j);
        end
    end
end
toc;
%%
if w==1
    %% normalization 1
    tic;
    C = B-repmat(mean(B),m,1);
    d = std(B);
    D = repmat(d,m,1);
    i=1;
    counter=1;
    t=n;
    while i <= t
        if d(counter)==0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C=C./D;
    toc;
end
```

```
Record = zeros(20000,1);
order=1;
COUNTER=1;
%j=1;
%% deleting the >3 features
tic;
for i=1:n
    j=1;
    COUNTER=1;
    while j <= m
        if C(j,i) > 3
            Record(order)=A(COUNTER,h);
            C(j,:)=[];
            A(COUNTER,:)=[];
            X(COUNTER)=[];
            m=m-1;
            order=order+1;
        else
            j=j+1;
            COUNTER=COUNTER+1;
        end
    end
end
toc;
%%result1
C = [C,A(:,(h-1):h)];
[m n] = size(C);
xlswrite('Normalization_Imputation_Screening_filtered.xlsx', C);
xlswrite('list_of_deletingSNPs.xlsx', Record);
xlswrite('output.labels.xlsx',X);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% read in the file
%% normalization with the imputation
function normalization_merge
w = input('Normalization method number: ')
A = xlsread('real.emerged.xlsx');
[m h] = size(A) %% m=808 n=65 807*64
n=h-2;
B = A(:,1:n);
%% imputation procedure for average
tic;
B(find(isnan(B)))=0;
AVG = mean(B);
%size(AVG)
for i=1:m
    for j=1:n
        if B(i,j)==0
            B(i,j) = AVG(j);
        end
    end
end
toc;
%%
if w==1
    %% normalization 1
    tic;
    C = B-repmat(mean(B),m,1);
    d = std(B);
    D = repmat(d,m,1);
    i=1;
    counter=1;
    t=n;
    while i <= t
        if d(counter)==0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C=C./D;
    toc;

elseif w == 2
    %% normalization 2
    tic;
    C=B;
```

```
    i=1;
    t=n;
    counter=1;
    d = max(abs(B));
    D = repmat(d,m,1);
    while i <= t
        if d(counter) == 0
            C(:,i)=[];
            D(:,i)=[];
            t=t-1;
        else
            i=i+1;
        end
        counter=counter+1;
    end
    C = C./D;
    toc;
end
%%result
%C = [C,A(:,(h-1):h)];
xlswrite('Normalization_Imputation_Screening_1.xlsx', C);
```

Appendix B

Supplementary Material

B.1 Some Breast Cancer Driver Genes

NOTCH2
hN2
AGS2
RHOC
H9
ARH9
ARHC
RHOH9
gstm1
mu
h-b
gst1
gth4
gtm1
mu-1
gstm1-1
gstm1a-1a
gstm1b-1b
rap1a
ap1
krev1
krev-1
mgp21
tacstd2
egp1
gp50
m1s1
egp-1
trop2
ga733-1
ga733-1
muc1
ema

```
em
pum
kl-6
mam6
pemt
cd227
h23ag
mckd1
muc-1
ca 15-3
muc-1/x
muc1/zd
muc-1/sec
parp1
parp
PARP
PPOL
ADPRT
ARTD1
ADPRT1
PARP-1
ADPRT 1
pADPRT-1
MAGI3
AKT3
MAGI3-AKT3
HER2
CBFB
RUNX1
```

A list of breast cancer driver genes compiled from PubMed.

B.2 SAMtools and BCFtools

SAMtools provides utilities for manipulating alignments in BAM/SAM format. It allows users to view their alignments, sort and index them for quicker manipulation, and merge multiple BAM/SAM files into one. SAMtools can also generate BCF files from BAM files, where each line of the file represents a genomic position and contains information about that position.

BCFtools allows users to convert between BCF and VCF format, call variant allele candidates, estimate allele frequencies, and index sorted BCFs for random access.

B.3 Genome Analysis Toolkit (GATK)

GATK is a software packaged developed by the Broad Institute for the analysis of next-generation sequencing data. We use its variant discovery and genotyping capabilities.

Selected Bibliography Including Cited Works

[1] About BGI. `http://www.genomics.cn/en/navigation/show_navigation?nid=4095`. Accessed: 6-23-2014.

[2] Rohit Chandra. *Parallel programming in OpenMP*. Morgan Kaufmann, 2001.

[3] Alexis Christoforides and John D Carpten et al. *Identification of Somatic Mutations in Cancer through Bayesian-based Analysis of Sequenced Genome Pairs*, volume 14. BMC Genomics, 2013.

[4] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[5] Ding J, Bashashati A, Roth A, Oloumi A, Tse K, Zeng T, Haffari G, Hirst M, Marra MA, Condon A, Aparicio S, and Shah SP. Feature-cased classifiers for somatic mutation detection in tumor-normal paired sequencing data. *Bioinformatics*, 28:167–75, 2012.

[6] Kensuke Nakamura, Taku Oshima, Takuya Morimoto, Shun Ikeda, Hirofumi Yoshikawa, Yuh Shiwa, Shu Ishikawa, Margaret C. Linak, Aki Hirai, Hiroki Takahasi, Md. Altaf-Ul-Amin, Naotake Ogasawara, and Shigehijo Kanaya. Sequence-specific error profile of illumina sequencers. *Nucleic Acids Research*, 39, 2011.

[7] Su Yeon Kim and Terence Speed. *Comparing Somatic Mutation-Callers: beyond Venn Diagrams*, volume 14. BMC Informatics, 2013.

[8] Heng Li, Jue Ruan, and Richard Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *CSH Press Genome Research*, 18:1851–1858, 2008.

[9] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Nicola D Roberts and R Daniel Kortschak et al. *A Comparative Analysis of Algorithms for Sometic SNV Detection in Cancer*, volume 29. Oxford University Press, 2013.

[11] Yuchi Shirashi and Yusuke Sato et al. *An Empirical Bayesian Framework for Somatic Mutation Detection from Cancer Genome Sequencing Data*, volume 41. Nucleid Acid Research, 2013.

[12] Julie Sleep, Andreas Schreiber, and Ute Baumann. Sequencing error correction without a reference genome. *BMC Bioinformatics*, 14:367, 2013.

[13] Xindong Wu and Vipin Kumar et al. *Top 10 Algorithms in Data Mining*, volume 14. Springer, 2008.

[14] Yan Guo, Jiang Li, Chung-I Li, Jirong Longand David C Samuels, and Yu Shyr. The effect of strand bias in illumina short-read sequencing data. *BMC Genomics*, 13:666, 2012.

[15] Yen-Chun Chen, Tsunglin Liu, Chun-Hui Yu, Tzen-Yuk Chiang, and Chi-Chuan Hwang. Effects of gc bias in next-generation-sequencing data on *De Novo* genome assembly. *PLoS ONE*, 8, 2013.