IndCh Paper - Variant Analysis (Strategy and Log)

By Tejashwini Alalamath

Candidate SNPs

Strategy

1. Aim : To find SNPs that characterize the 2Rb region.
2. Known information :
   UCI is homozygous inverted form, STE2 is heterozygous, IndCh is hypothesized to be homozygous uninverted (Standard)
3. The central dataset used here is a Matrix file. From the VCF file, a matrix is generated expressing the genotype of every sample, for a given chromosomal position.
   [0=Homozygous reference, 1 = Heterozygous alternate, 2 = Homozygous alternate]

Commands used - (Big codes attached below)

1. **Step-1**

i) Raw data download and VCF processing (Script attached below)

Ste2

fastq-dump --split-files SRR1168951 &

UCI

fastq-dump --split-files SRR11672504

[IndCh and population samples are in-house]

ii) Merging the VCF file (For Candidate SNPs)

Only merge the IndCh, STE2 and UCI vcf files.

bcftools merge -m id IndCh.vcf.gz STE2.vcf.gz UCI.vcf.gz -o IndCh_STE2_UCI_SNP.vcf.gz -O z

2. **Step-2**

Convert the Merged VCF file into a matrix file. *(Code below)*

3. **Step-3**
   Getting the Candidate SNPs

a) The file looks like this

Chrom.Pos	IndCh	STE2	UCI
chr2.17	2	0	2
chr2.110	0	0	1
chr2.121	1	0	1
chr2.128	0	1	0
chr2.238	2	1	0

b) We need columns which are 2 for IndCh, 1 for STE2 and 0 for UCI, hence we extract those rows only using the awk command.

-> We first copy the header in the other file
head -n1 IndCh_STE2_UCI_SNP_012.tsv > Candidate_SNP_ISU_Full_012.tsv

-> Filter it with awk
awk '($2==2 && $3==1 && $4==0)' IndCh_STE2_UCI_SNP_012.tsv >> Candidate_SNP_ISU_Full_012.tsv

-> Final Output

Chrom.Pos	IndCh	STE2	UCI
chr2.23707	2	1	0
chr2.24885	2	1	0
chr2.27398	2	1	0
chr2.33214	2	1	0
chr2.33895	2	1	0
chr2.40818	2	1	0
chr2.42653	2	1	0

-> Getting only SNP position list
cut -f1 Candidate_SNP_ISU_Full_012.tsv > Candidate_SNP_ISU_Full_list.txt

#Note, full chr2 candidate SNP list allows us to explore non-2Rb regions as well.

The above list is used to extract the genotype status of population samples at those positions of interest.

#The VCF files (Merged VCF of population samples) are filtered for the 2Rb region
-> bcftools view -r chr2:55256000-71808800 ${F} > "${F%_uci2_merged_snp_filtered.vcf.gz}_uci2_merged_SNP_Chr_2Rb_filtered.vcf"

#Convert vcf to a matrix file (Code below) and use grep to get positions of interest.
-> grep -Fwf Candidate_SNP_ISU_Full_list.txt Lab_uci2_merged_SNP_Chr_2Rb_012.tsv > Output_Candidate_SNP.tsv

----
The above output file is then used to plot heatmaps/PCA/tSNE/KMeans,etc or carry out DESeq filtering and get significant SNPs among the obtained set.

DESeq Filtering (All codes attached below)

1. In order to filter the sample cluster-wise, we needed to find which samples belong to which **cluster**, for which a **neighbour joining tree** *(using the code below)* was constructed and the samples and cluster names were manually noted down.

2. The following steps were carried out to re-order the file as input for DESeq

   cut -f1,3,8,14,17,20,22,26,27,28,36,37,38,40,41,48,52,57,60 Candidate_Lab_uci2_2rb_snp_012.tsv > ste2.tsv

   cut --complement -f1,3,8,14,17,20,22,26,27,28,36,37,38,40,41,48,52,57,60 Candidate_Lab_uci2_2rb_snp_012.tsv > indch_snp.tsv

   paste ste2_snp.tsv indch_snp.tsv > Reordered_Candidate_Lab_uci2_2rb_snp_012.tsv

3. The above file was then used as an input for DESeq. *(The code is given below)*

#Note: Here, the wild samples becomes our test set

Scripts used

---

1. VCF file generation

[The following script was modified as per the samples]

#Pipeline for generating vcf file
#1.Get the reference file ready
#bowtie2-build UCI2_chr2.fa UCI2_chr2_idx

#2.Map your reads to it
#bowtie2 -x UCI2_chr2_idx -1 SRR1168951_1.fastq -2 SRR1168951_2.fastq -S SRR1168951_STE2_Illumina_against_UCI2_Chr2_.sam

#3.Convert it to bam file
samtools view -bS SRR1168951_STE2_Illumina_against_UCI2_Chr2_.sam -o SRR1168951_STE2_Illumina_against_UCI2_Chr2_.bam

#4.Sort the bam file
samtools sort SRR1168951_STE2_Illumina_against_UCI2_Chr2_.bam -o SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted.bam

#5.Index it
samtools index SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted.bam

#6.Get the reference ready
#samtools faidx UCI2_chr2.fa

#7.Penultimate step before vcf
samtools mpileup -vu -f UCI2_chr2.fa -o SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted.mpileupvcf SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted.bam

#8.Finally call the vcf

```
bcftools call -vmO v -o
SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted_mpileup.call.vcf
SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted.mpileupvcf
```

#For further processing

```
bgzip SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted_mpileup.call.vcf
tabix SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted_mpileup.call.vcf.gz

bcftools filter -i "DP>3 && QUAL>10"
SRR1168951_STE2_Illumina_against_UCI2_Chr2_sorted_mpileup.call.vcf.gz -o
STE2_dp3_qual10.vcf.gz -O z
```

#VCF Processing

#SNP filtering

```
#!/bin/bash
for F in ~/File_Path/*.vcf.gz
do
bcftools filter -i "TYPE='SNP'" ${F} -o "${F%_dp3_qual10.vcf.gz}_snp_filtered.vcf.gz" -O
z
Done
```

#Indel filtering

```
for F in ~/File_Path/*.vcf.gz
do
bcftools filter -i "TYPE='INDEL'" ${F} -o "${F%_dp3_qual10.vcf.gz}_indel_filtered.vcf.gz"
-O z
done
```

#Merging the files

bcftools merge -m id File1.vcf.gz File2.vcf.gz File.vcf.gz -o Merged.vcf.gz -O z

---

2. Matrix_012.py

#Python code to make a 0,1,2 matrix

#Importing all the packages required to extract variants from the vcf file

```
import numpy as np
import pandas as pd
import allel; print('scikit-allel', allel.__version__)
import os

print("Loading vcf file")
```

```
#Note: The scikit-allel has some inbuilt functions to analyse variant files

#Now, we input the vcf file (bgzipped/indexed or just vcf) over here
#The callset object returned by read_vcf() is a Python dictionary (dict). It contains several NumPy arrays, each of which can be accessed via a key.
callset = allel.read_vcf('input.vcf.gz', fields='*')

#Here, we are inputting the Chrom and Pos into a list (Revise python) using the callset key of variants/CHROM and variants/POS
chrom = callset['variants/CHROM']
pos = callset['variants/POS']

#Later- Read numpy array vs others. Note numpy arrays help in large data manipulation
#Now, we are creating a genotype array using the key 'calldata/GT'.
allel.GenotypeArray is the inbuilt function telling its a genotype array
gt = allel.GenotypeArray(callset['calldata/GT'])
#Now, in this array whatever is equal to -1, change it to 0 (Reference) [It will be -1 if there was no call found at that position]
gt[gt == -1] = 0

#[This genotype array is our file, hence gt.shape gives the information / structure of the file ]
print("Structure of VCF File : ",gt.shape)

#What remains is assigning the names. For this, we create a blank list called cp and from the previously created lists, we merge both the columns (chrom and pos)
cp = []
for i in range(len(chrom)):
        tmp=chrom[i] + '.' + str(pos[i])
        cp.append(tmp)

#The genotype array is in the format 0/0, 1/1, etc. These step resolves it to a single digit
#Transform the genotype data into a 2-dimensional matrix where each cell has the number of non-reference alleles per call
gt_012 = gt.to_n_alt(fill=0)  #converting to 012

#New array where you convert it to a data frame
gt_012_df = pd.DataFrame(gt_012)

print("DataFrame built, Shuffling columns")

#We now add the names in proper order.
gt_012_df['Chrom.Pos'] = cp
cols = list(gt_012_df.columns)
cols = [cols[-1]] + cols[:-1]
gt_012_df = gt_012_df[cols]

print("Writing onto an output file 012.csv")
```

```
###Change output file name
gt_012_df.to_csv("output.tsv", index=None, sep='\t')

#manually changed header using shell or use system call

os.system("sed -i "1s/.*/Chrom.Pos\tIndCh\tSTE2\tUCI/" output.tsv")
```

---

3. PCA.py

```
#!/usr/bin/python
import pandas as pd
import os

df=pd.read_csv("TEST_sample.tsv",sep="\t") #Input file
df_t=df.transpose() #Transposing

df_t.to_csv("python_trans.tsv",sep="\t") #Writing onto another file

#--------------------------------Linux commands-------------------------------------------
os.system("sed -i '1d' python_trans.tsv") #The column header maybe repeated, hence the step

#Removing first column which had individual sample lanes
os.system('cut -f2- python_trans.tsv > no_samples.tsv') # no_samples.tsv is the transposed file with no sample names

#Adding a column of population names to the beginning of the file
os.system('paste Populations.txt no_samples.tsv > final.tsv') #Pasting the files

#Example of Populations.txt [This file has to be prepared as per the samples]
#Populations
#TI
#TI
#TII
#TII
#TII
#.
#.
#.
#.
#.

#Adding SNP names to a file #The first row will have the names of the SNPs
os.system("head -1 no_samples.tsv > features.tsv")
```

#2D PCA

```
import matplotlib.pyplot as plt
import pandas as pd

feat=pd.read_csv("features.tsv",sep="\t")
snp=[] #Empty list
print("Step-1")
for col in feat.columns:
        snp.append(str(col)) #Time consuming step
df = pd.read_csv("final.tsv",sep="\t") #Reading into a dataframe
from sklearn.preprocessing import StandardScaler
features=snp
x = df.loc[:, features].values
y = df.loc[:,['Populations']].values
x = StandardScaler().fit_transform(x)
print("Step-2")
from sklearn.decomposition import PCA #PCA calculations
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component
1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['Populations']]], axis = 1)
variance=pca.explained_variance_ratio_ #To get variance on the x and y axis, note down
print(variance)
finalDf.to_csv("final_df.tsv",sep="\t")
```

#Visualization done on the local system by inputing the variance values obtained from the above output

```
import matplotlib.pyplot as plt
import pandas as pd

finalDf=pd.read_csv("final_df_2L_deseq_all.tsv",sep="\t")
variance=[0.33548281,0.10954588]
 #Enter values found on server
pc1="Principal Component 1 ("+str(round((variance[0]*100),2))+"%)"
pc2="Principal Component 2 ("+str(round((variance[1]*100),2))+"%)"
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel(pc1, fontsize = 17)
ax.set_ylabel(pc2, fontsize = 17)
ax.set_ylim([-10, 100])
ax.set_title('2 component PCA', fontsize = 20)
```

```
targets = ['B','M','TI','TII','TIII','TIV','I', 'S', 'U']
colors = ['r', 'b','c','m','y','lime','maroon','k','navy']
for target, color in zip(targets,colors):
        indicesToKeep = finalDf['Populations'] == target
        ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
        , finalDf.loc[indicesToKeep, 'principal component 2']
        , c = color
        , s = 50)
ax.legend(targets)
ax.grid()
plt.show()
```

---

4. Make_tree.r

```
library(ape)
#creating unrooted NJ tree
data2=read.table("Candidate_Lab_uci2_2rb_snp_012.tsv",header=T)
rownames(data2)=data2$Chrom.Pos
data2$Chrom.Pos=NULL
data2=t(data2)
stree=nj(dist.gene(data2))
write.tree(phy=stree, file="Candidate_Lab_uci2_2rb_snp.newick")
```

---

5. DESeq.r [3781 SNPs]

```
library(DESeq2)
library(DESeq)

#put 012 tsv file path/name
print("Step 1 - Reading the table and arrangement")
data=read.table("Reordered_Candidate_Lab_uci2_2rb_indel_012.tsv",header=T,sep="\t")
rownames(data)=data$Chrom.Pos
data$Chrom.Pos=NULL
##change according to population name and size

print("Step 2 - Defining the groups")
#UCI2.0 reference
group=c(rep('S',19),rep('I',43))

print("Step 3 - Putting it in CountDataSet")
countDataSet<- newCountDataSet(data,group)
countDataSet<-estimateSizeFactors(countDataSet)
countDataSet<-estimateDispersions(countDataSet,fitType="local")
```

```
###for each population combination and p-value of choice
print("Step 4 - Generating the output for negative binomial test")

DEVal=nbinomTest(countDataSet,"I","S")
p1=subset(DEVal,pval<0.0005)
write.table(p1,"I_S_0005.txt",quote=F)
```

---

6. Prep.sh [3781 SNPs]

```
#Script
cat I_S_0005.txt | grep -v "baseMean" | sort -u -k1 -n | cut -f2 -d " " > I_S_pval_0005.txt

head -n1 Candidate_Lab_uci2_2rb_indel_012.tsv >>
Lab_Candidate_indel_2rb_DESeq_filtered_pval_0005.tsv
#Note here the txt file only has the chromosomal positions, hence we can extract the
positions from any master file. Here, the main master file is used instead of the
re-ordered one, to ease further downstream processing of plots.
grep -Fwf I_S_pval_0005.txt Candidate_Lab_uci2_2rb_indel_012.tsv >>
Lab_Candidate_indel_2rb_DESeq_filtered_pval_0005.tsv
```

---

7. SNP_Block.py [277 SNPs]

```
#!/usr/bin/python

import numpy as np
import pandas as pd
import os

#To write later
os.system('cut -f1 Lab_samples.tsv > Pos.tsv')
os.system('cut -f2 -d "." Pos.tsv > Pos_3781.tsv)

df=pd.read_csv("Pos_3781.tsv",sep="\t")
pos1 = df['Pos'].tolist() #OR pos1 = df['Position'].tolist(),have to do edit to header in this case
mat=[]
test=[]
z=len(pos1)-1

#Code still has a certain degree of redundancy

for a in range(z):
    print("Iteration="+ str(a+1))
    ie1=pos1[a]
```

```
    m=ie1 + 1500
    print(m)
    res = [x for x in range(z) if pos1[x]<m]
    g=res[-1]
    test=pos1[a:g]
    if len(test)>6:
            mat.append(test)
            #print(mat)
    else:
            print("Moving on")


print("Final Matrix")
print(mat)

#Mat_df=pd.DataFrame(mat)
#Mat_df.to_csv("Pos_3781_SNP_Blocks.tsv", index=None, sep='\t')
#print(Mat_df)

#Now to remove redundancy [Help: Stackoverflow]

result = []

for d in mat:
        d = set(d)

        matched = [d]
        unmatched = []
        # first divide into matching and non-matching groups
        for g in result:
        if d & g:
        matched.append(g)
        else:
        unmatched.append(g)
        # then combine all matching groups into one group
        # while leaving unmatched groups intact
        result = unmatched + [set().union(*matched)]

print(result)
```

8. Annotate.sh [1740 exonic SNPs]

java -jar /File_Path/snpEff.jar eff -c /File_Path/snpEff.config -v anstephv2 UCIwg_IndChStdwg.call.vcf > UCIwg_IndChStdwg_annotated_anstephv2.vcf

java -jar /File_Path/SnpSift.jar filter "(QUAL>=30)" UCIwg_IndChStdwg_annotated_anstephv2.vcf > UCIwg_IndChStdwg_annotated_anstephv2_qual30.vcf

java -jar /File_Path/SnpSift.jar filter "ANN[*].EFFECT has 'missense_variant'" UCIwg_IndChStdwg_annotated_anstephv2_qual30.vcf > UCIwg_IndChStdwg_annotated_anstephv2_qual30_missense.vcf

java -jar /File_Path/SnpSift.jar filter "ANN[*].EFFECT has 'synonymous_variant'" UCIwg_IndChStdwg_annotated_anstephv2_qual30.vcf > UCIwg_IndChStdwg_annotated_anstephv2_qual30_synonymous.vcf

9. Prep.sh [1740 exonic SNPs]

 # For a readable output
#Now the 2Rb annotated file has been filtered for missense and synonymous variants, however they are two separate files and therefore we need to merge them. Here, we have extracted the positions from those two files. We will merge and sort them into one position file, using this we will extract the Candidate Annotated missense and synonymous variants from the master file and write them to another file.

cat Candidate_synonymous_variant.txt Candidate_missense_variant.txt | sort -n | uniq > Synonymous_Missense_Position_Combined.txt

grep -Fwf Synonymous_Missense_Position_Combined.txt Chr2Rb_Candidate_Annotated.vcf > Chr2Rb_Candidate_Annotated_Synonymous_Missense.vcf

cut -f4 -d "|" Chr2Rb_Candidate_Annotated_Synonymous_Missense.vcf | uniq > Chr2Rb_Candidate_Annotated_Synonymous_Missense_Geneid_list.txt

#The positions here correspond to the exonic Candidate SNPs. This position list can now be further used for extracting the population wide status from the 012 matrix '.tsv' file and generating a plot.
