Q1.a

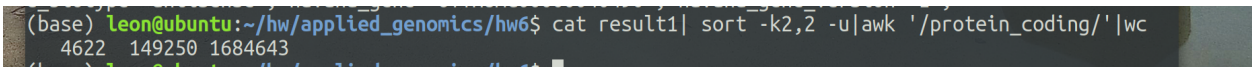
R code:

```
library(openxlsx)
a = c('Chromosome','Start','End')
b = read.xlsx('npjgenmed201627-s3.xlsx',sheet=4,startRow = 2)
c = b[a]
d = cbind(c[1],c[2],c[3]+1)
write.table(d,'q1.bed',sep='\t',row.names = F,col.names = F,quote = F)
```

Upload to https://genome.ucsc.edu/cgi-bin/hgLiftOver, download the bed file

Delete the commented line and then shell code:

```
sed 's/^/chr/' Homo_sapiens.GRCh38.87.gtf > Homo_sapiens.GRCh38.87_result.gtf
bedtools intersect -a hglft_genome_5706_29fbb0.bed -b Homo_sapiens.GRCh38.87_result.gtf -wa -wb
  > result1
cat result1| sort -k2,2 -u|awk '/protein_coding/'|wc
```

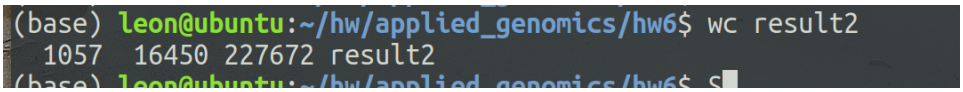


So we have 4622 variants in protein coding genes.

Q1.b

Similar process as shown in 1a, preprocess the homo_sapiens file (sed 's/^/chr/') and then:

```
bedtools intersect -a hglft_genome_5706_29fbb0.bed
  -b homo_sapiens.GRCh38.Regulatory_Build.regulatory_features.20161111_result.gtf -wa -wb
  > result2
wc result2
```



So 1057 variants are in annotated regulatory regions.

Q1.c

```
leon@ubuntu:~/hw/applied_genomics/hw6$ cat result2 | tr '=' '\t'|cut -f 16|tr ';' '\t'|cut -f 1| sor
t|uniq -c
    241 CTCF binding site
    303 Open chromatin region
    100 Predicted enhancer region
     93 Predicted promoter
    274 Predicted promoter flanking region
     46 Transcription factor binding site
```

So open chromatin region has the most variants.

Q1.d

Essentially we need to test if those variants in annotated regulatory region are uniformly distributed, so we use chi-square test:

```r
chisq.test(c(241,303,100,93,274,46))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  c(241, 303, 100, 93, 274, 46)
## X-squared = 337.88, df = 5, p-value < 2.2e-16
```

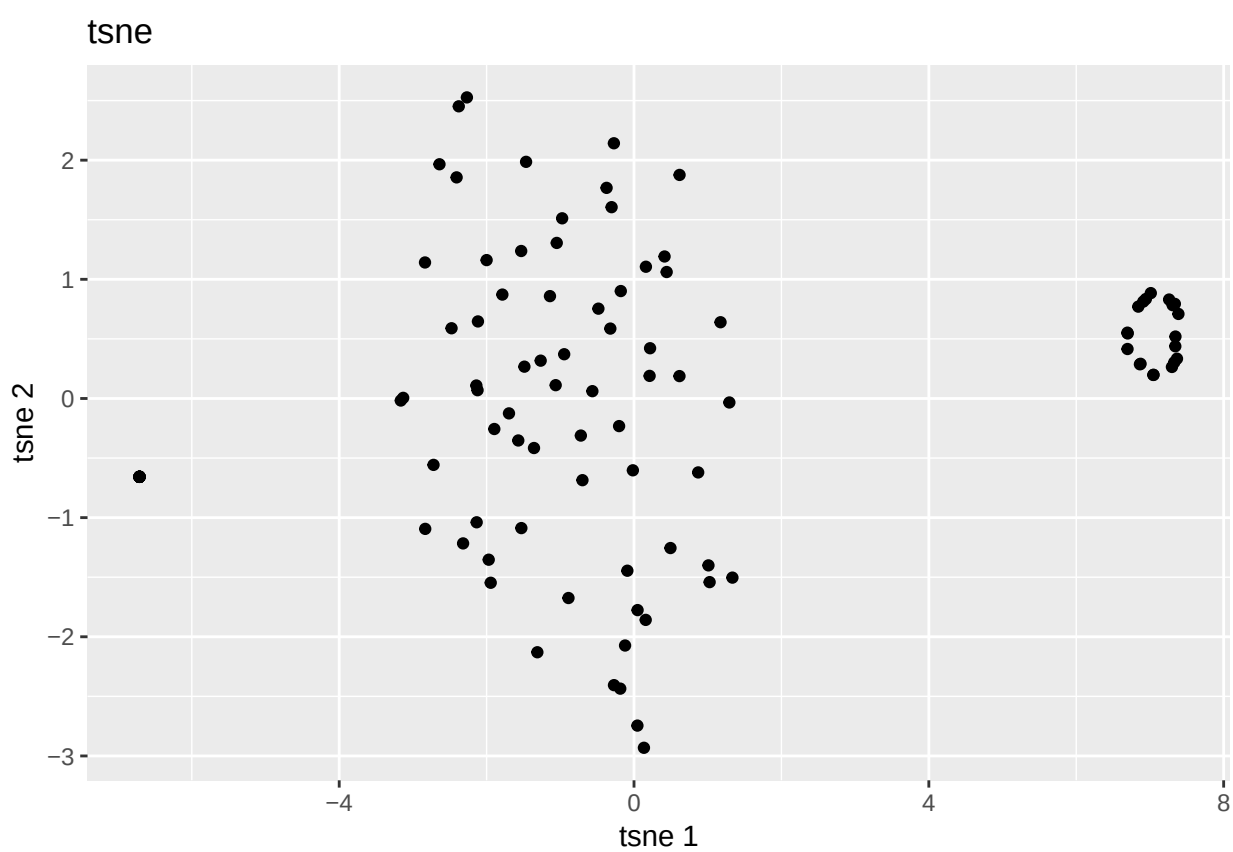
p-value < 0.05, meaning they are not randomly distributed. In this case, it's enriched.

Q2.a

```r
#tsne
library(Rtsne)
library(ggplot2)
library(plotly)
```

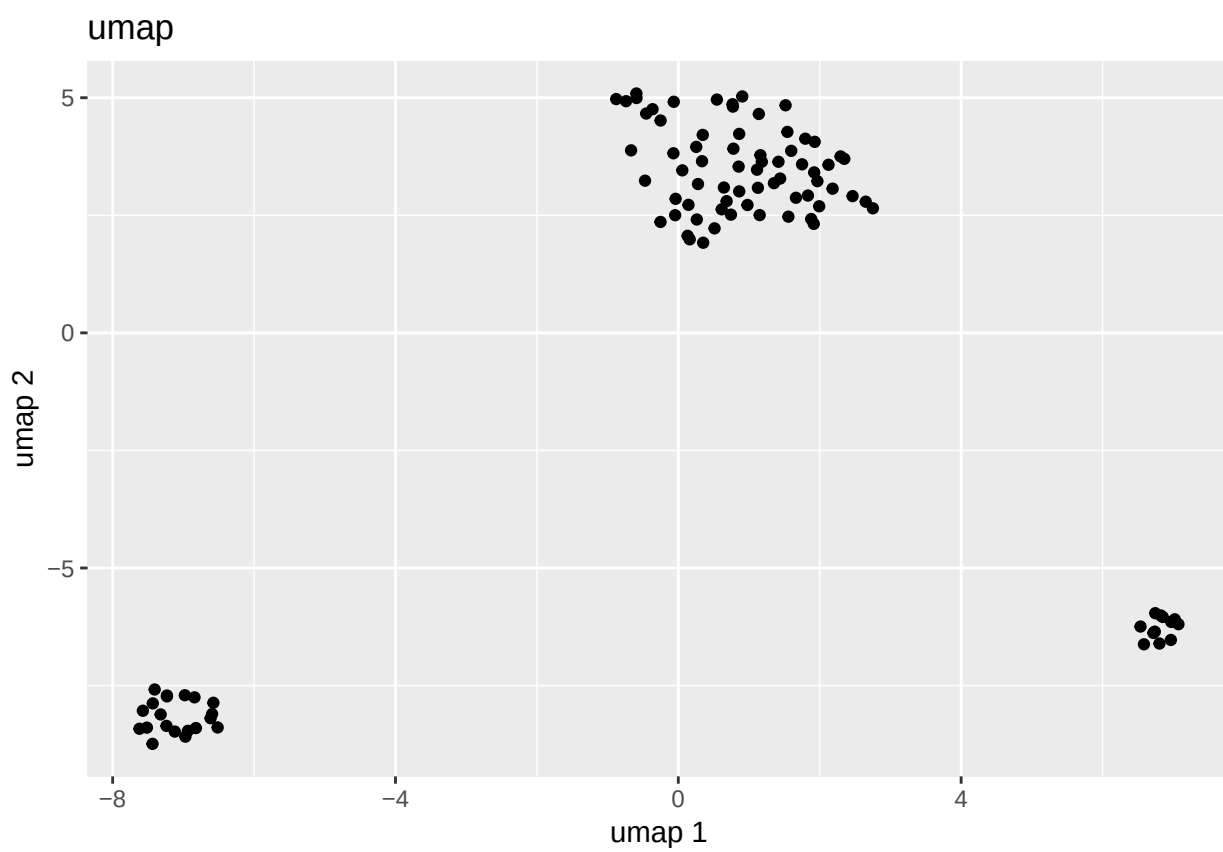
```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```r
set.seed(101)
dat = as.matrix(read.table('expression.txt',row.names = 1,header = T))
tsne = Rtsne(dat)
# plot(tsne$Y, asp =1 ,main="tSNE", xlab="tSNE dimension 1", ylab="tSNE dimension 2")
ggplot(as.data.frame(tsne$Y),aes(x=V1,y=V2))+
  geom_point()+labs(x= 'tsne 1', y = 'tsne 2', title = 'tsne')
```



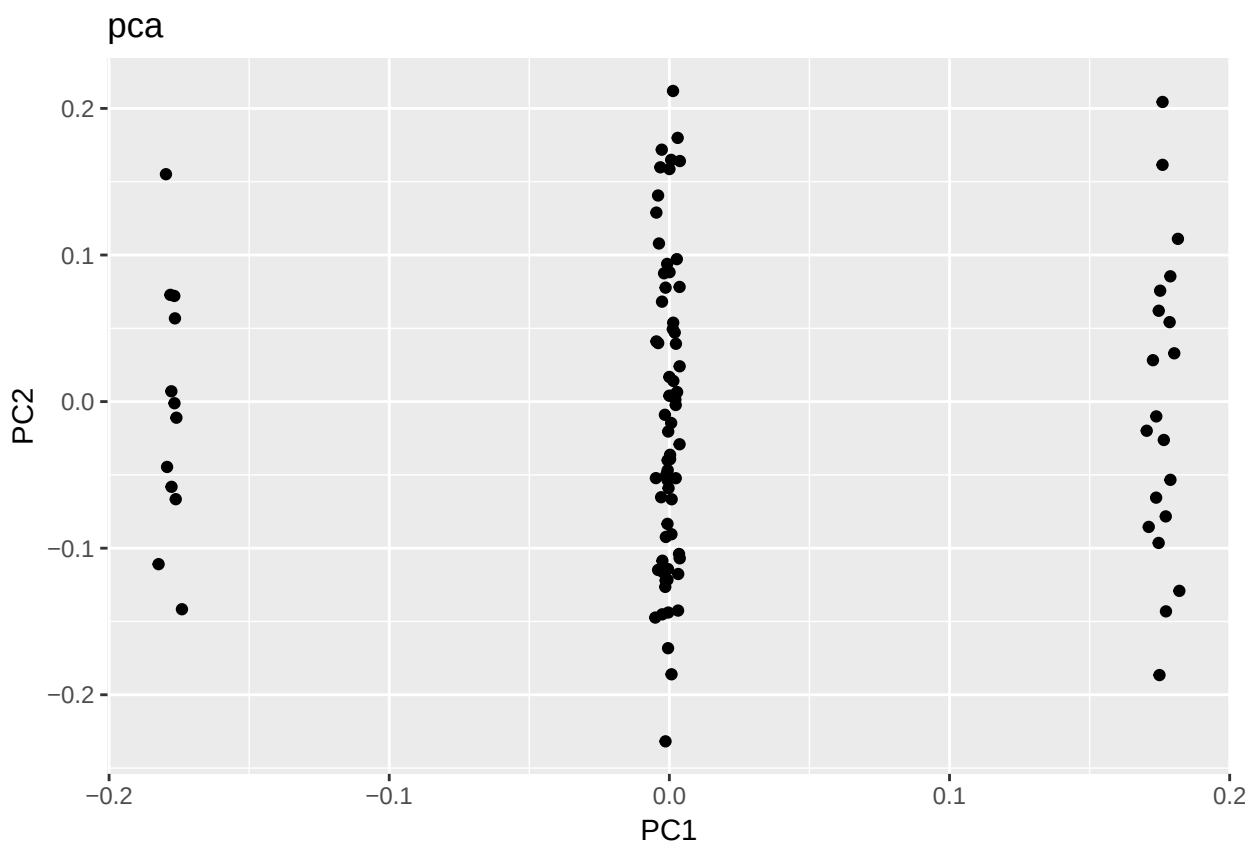
Q2.b

```
library(umap)
library(plotly)
set.seed(102)
dat = as.matrix(read.table('expression.txt',row.names = 1,header = T))
umap = umap(dat)
ggplot(as.data.frame(umap$layout),aes(x = V1,y=V2))+
  geom_point()+labs(x= 'umap 1', y = 'umap 2', title = 'umap')
```



Q2.c

```
#pca
library(plotly)
set.seed(101)
dat = as.matrix(read.table('expression.txt',row.names = 1,header = T))
pca = prcomp(t(dat))
ggplot(as.data.frame(pca$rotation),aes(x=PC1,y=PC2))+
  geom_point()+labs(title = 'pca')
```




```
#time comparison and heatmap
ptm <- proc.time()
tsne = Rtsne(dat)
proc.time() - ptm
```

```
##    user  system elapsed
##    0.24    0.04    0.28
```

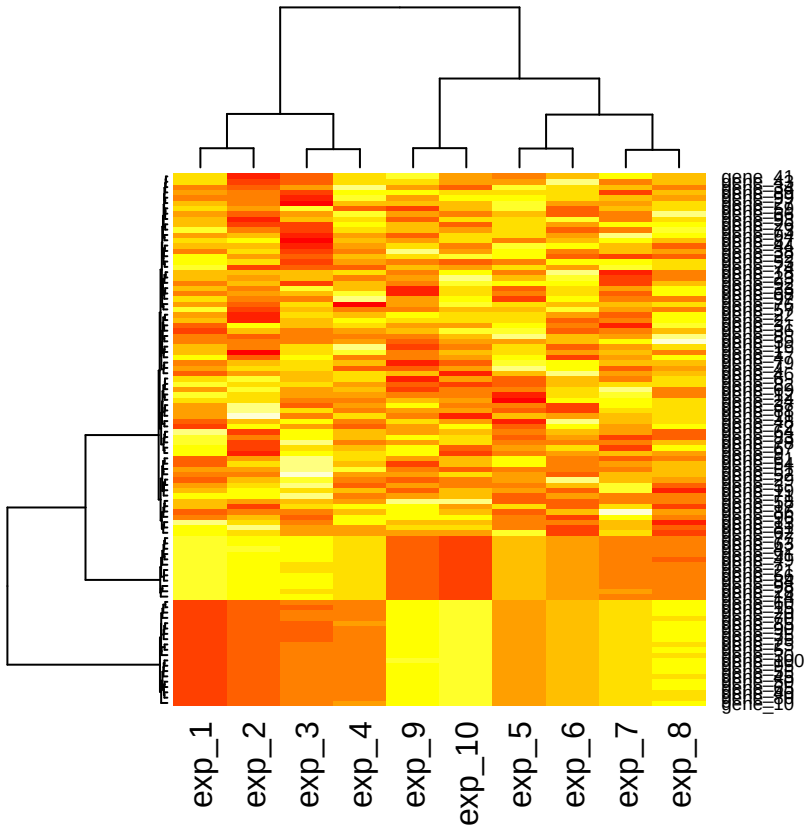
```
ptm <- proc.time()
umap = umap(dat)
proc.time() - ptm
```

```
##    user  system elapsed
##    0.45    0.00    0.47
```

```
ptm <- proc.time()
pca = prcomp(t(dat))
proc.time() - ptm
```

```
##    user  system elapsed
##       0       0       0
```

```
ptm <- proc.time()
heatmap(dat)
```




```
proc.time() - ptm
```

```
##    user  system elapsed
##    0.09    0.00    0.09
```

From results shown above, PCA is the fastest and umap is the slowest. Heatmap is pretty easy to understand and straightforward. For t-sne and umap, the relative positions of clusters are not fixed.

Q3.a

```
s1 = [1]
s2 = [0]
seq = 'CAACATTGTCGCCATTGCTCAGGGATCTTCTGAACGCTC'
dic1 = {'A':0.15,'C':0.35,'G':0.35,'T':0.15}
dic2 = {'A':0.3,'C':0.2,'G':0.15,'T':0.35}
#forward
for i in range(len(seq)):
    s1.append(float(s1[i])*0.5*dic1[seq[i]] + float(s2[i])*0.25*dic2[seq[i]])
    s2.append(float(s1[i])*0.5*dic1[seq[i]] + float(s2[i])*0.75*dic2[seq[i]])
print(s1)
```

```
## [1, 0.175, 0.02625, 0.005906249999999999, 0.0017226562499999998, 0.00036175781249999994, 9.948339843
```

```
print(s2)
```

```
## [0, 0.175, 0.05249999999999999, 0.013781249999999998, 0.00310078125, 0.0008268749999999999, 0.000244
```

Trellis:

```
[1, 0.175, 0.02625, 0.005906249999999999, 0.0017226562499999998, 0.00036175781249999994, 9.948339843749999e-05, 2.8827575683593745e-05,
7.728333892822263e-06, 1.7254681835174555e-06, 5.028146554470061e-07, 1.2191244353234764e-07, 3.0822287677690373e-08, 7.883775733433288e-09,
1.5560476684906142e-09, 4.2169153161427027e-10, 1.2189776627087692e-10, 3.2673592937546844e-11, 8.485706794962378e-12, 1.8632972595874304e-12,
5.419288084036971e-13, 1.1366708945693907e-13, 2.9630938671599184e-14, 7.02701431896244e-15, 1.6313605466369622e-15, 3.049490381183246e-16,
8.150869319032442e-17, 2.4203207500968003e-17, 5.672378802623015e-18, 1.5967607474350624e-18, 4.7640440189672265e-19, 1.118856877499451e-19,
2.948736046744015e-20, 5.909208836795987e-21, 1.441029845780094e-21, 4.240156336025607e-22, 1.0299097793487246e-22, 2.6051794240788738e-23,
5.63838185701073e-24, 1.6370856467225717e-24]
[0, 0.175, 0.05249999999999999, 0.013781249999999998, 0.00310078125, 0.0008268749999999999, 0.0002441865234374996, 7.156021728515624e-05,
1.3095350189208981e-05, 4.017154466629027e-06, 9.045301021099088e-07, 1.8975220119059078e-07, 4.979750779674945e-08, 1.2863526513108233e-08,
3.4855766454568487e-09, 1.0316674445692l9e-09, 3.024395690704902e-10, 5.535656061783361e-11, 1.4021362856745739e-11, 4.317035759517935e-12,
9.736323843554906e-13, 2.5971194711026263e-13, 4.910933470486876e-14, 1.0710214421827606e-14, 2.4346266282740328e-15, 6.701430323594294e-16,
1.9878372385322456e-16, 4.408157988629046e-17, 1.3386655282723844e-17, 3.939425421911735e-18, 8.70346944087896e-19, 2.641964029653269e-19,
4.930209068983967e-20, 1.3304522440271935e-20, 3.436708211820884e-21, 7.676864547846491e-22, 1.6056746204372114e-22, 4.2108540445160856e-23,
1.3007376434913879e-23, 2.9378232902139595e-24]
```

So probability = 1.6370856467225717e-24.

Q3.b

```
s1 = [1]
s2 = [0]
seq = 'CAACATTGTCGCCATTGCTCAGGGATCTTCTGAACGCTC'
dic1 = {'A':0.15,'C':0.35,'G':0.35,'T':0.15}
dic2 = {'A':0.3,'C':0.2,'G':0.15,'T':0.35}
#forward
for i in range(len(seq)):
    s1.append(float(s1[i])*0.5*dic1[seq[i]] + float(s2[i])*0.25*dic2[seq[i]])
    s2.append(float(s1[i])*0.5*dic1[seq[i]] + float(s2[i])*0.75*dic2[seq[i]])
#print(s1)
#print(s2)
#viterbi
s3 = []
s1 = [1]
s2 = [0]
for i in range(len(seq)):
    s12s1 = float(s1[i])*0.5*dic1[seq[i]]
    s22s1 = float(s2[i])*0.25*dic2[seq[i]]
    s12s2 = float(s1[i])*0.5*dic1[seq[i]]
    s22s2 = float(s2[i])*0.75*dic2[seq[i]]
    s1.append(max(s12s1,s22s1))
    s2.append(max(s12s2,s22s2))
    if s1[i] > s2[i]:
        s3.append('S1')
    else:
        s3.append('S2')
print(s1)
```

```
## [1, 0.175, 0.013125, 0.0029531249999999996, 0.0005167968749999999, 9.966796874999998e-05, 2.61628417
```

```
print(s2)
```

```
## [0, 0.175, 0.03937499999999999, 0.008859374999999997, 0.0013289062499999998, 0.00029900390624999995,
```

```
print(s3[1:])
```

```
## ['S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2
```

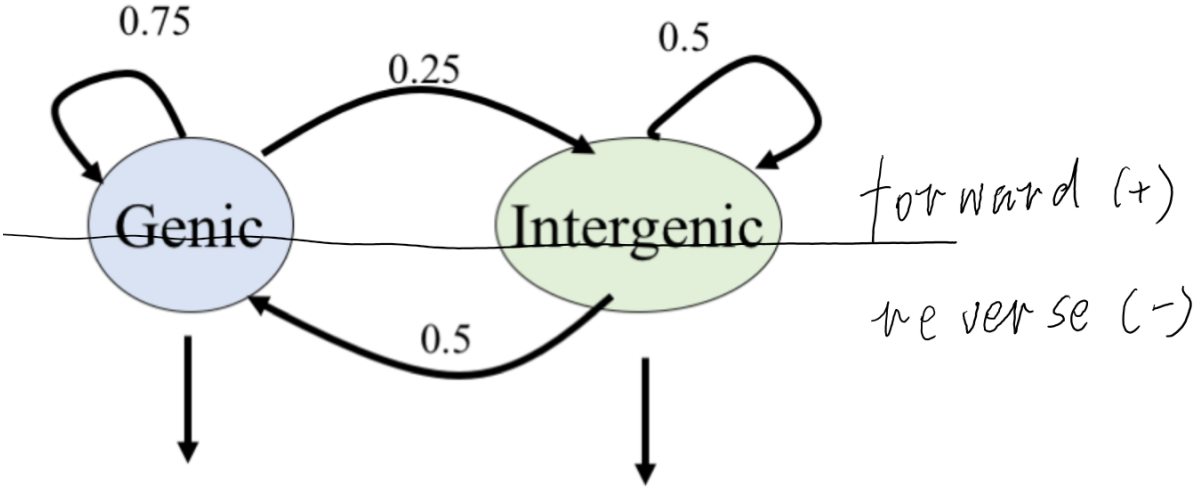
Trellis:

```
[1, 0.175, 0.013125, 0.0029531249999999996, 0.0005167968749999999, 9.966796874999998e-05, 2.6162841796874995e-05, 6.867745971679687e-06,
1.201855545043945e-06, 2.0281312322616574e-07, 3.5492296564579004e-08, 6.211151898801326e-09, 1.086951582290232e-09, 1.9021652690079058e-10,
1.7326261738110337e-11, 4.548143706253963e-12, 1.1938877228916652e-12, 2.089303515060414e-13, 3.656281151355724e-14, 5.2885495224966726e-15,
9.254961664369177e-16, 1.784885463842627e-16, 3.123549561724597e-17, 5.466211733018045e-18, 9.565870532781579e-19, 1.174402899586184e-20,
1.8832807611413732e-20, 3.295741331997403e-21, 7.415417996994156e-22, 1.9465472242109656e-22, 3.40645764236919e-23, 7.664529695330677e-24,
1.3412926966828683e-24, 1.9400840791305775e-25, 4.365189178043799 5e-26, 7.639081061576649e-27, 1.3368391857759134e-27, 2.3394685751078485e-28,
2.900463590567384e-29, 5.075811283492921 5e-30]
[0, 0.175, 0.03937499999999999, 0.008859374999999997, 0.0013289062499999998, 0.00029900390624999995, 7.848852539062499e-05, 2.0603237915039062e-
05, 2.3178642654418943e-06, 6.084393696784972e-07, 9.126590545177458e-08, 1.0267414363324641e-08, 1.5401121544986964e-09, 2.310168231748045e-10,
5.197878521433101e-11, 1.3644431118761889e-11, 3.5816631686749955e-12, 4.02937106475937e-13, 6.044056597139054e-14, 1.5865648567490017e-14,
2.3798472851235028e-15, 5.354656391527881e-16, 6.023988440468866e-17, 6.776986995527 4734e-18, 9.565870532781579e-19, 2.152320869875855e-19,
5.64984228342412e-20, 8.474763425136179e-21, 2.2246253990982467e-21, 5.839641672632897e-22, 8.759462508949346e-23, 2.2993589085992033e-23,
2.5867787721741036e-24, 5.820252237391732 5e-25, 1.3095567534131398e-25, 1.9643351301197097e-26, 2.2098770213846736e-27, 3.3148155320770103e-28,
8.701390771702151e-29, 1.305208615755323e-29]
['S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2',
'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2', 'S2']
```

In this case, S2 represents genic state.

Q3.c

Just split the model into two parts and assign emission probability and transition probability as needed for each strand(sort of symmetric):

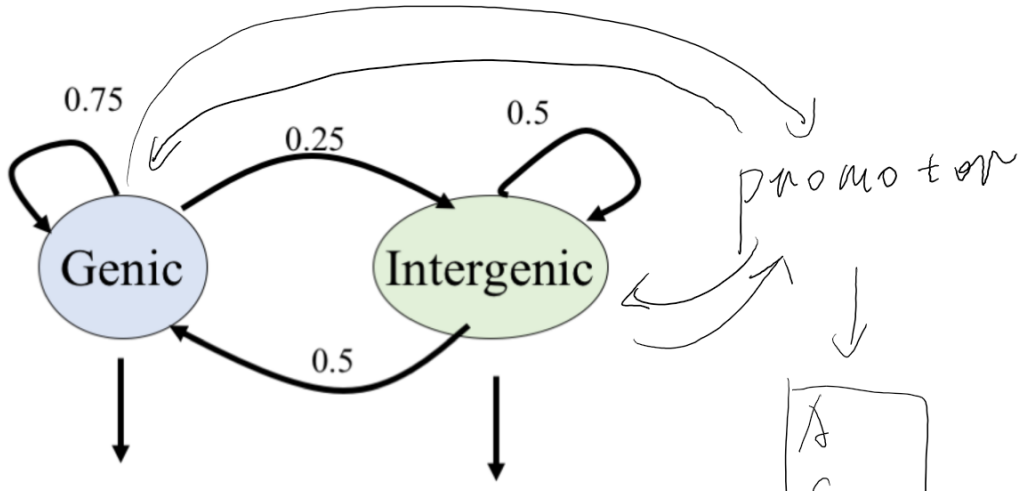


A	.30
C	.20
G	.15
T	.35

A	.15
C	.35
G	.35
T	.15

Q3.d

Add another state and assign emission probability and transition probability as shown below:



A	.30
C	.20
G	.15
T	.35

A	.15
C	.35
G	.35
T	.15

A
C
G
T