实验项目 3：真核生物基因组的基因分析和预测

一、分析平台：

1.1. 硬件平台：（硬件配置）：win10：CPU 1.70GHz　2.40GHz
Ubuntu：CPU 3.3GHz*2

1.2. 系统平台：（操作系统及其版本号）WIN10 专业版、Ubuntu【机房 s06@HP06】、IBM_S812L 小型机

1.3. 软件平台：（软件系统及其版本号，若是在线分析平台，还需要提供 URL 地址）R3.4.4、SUBLIME TEXT3

1.4. 数据资源：任课教师提供【AP-genome-draft】;
Uniprot 数据库中检索 taxonomy:fungi AND reviewed:yes 下载的所有真菌的蛋白序列

二、实验步骤：

1、基因组核酸序列的获取：
任课教师提供【AP-genome-draft】

2、创建本地 BLAST 数据库： 使用 makeblastdb 程序，对上述 FASTA 格式的基因组序列进行处理，建立本 地 BLAST 数据库。

```
#创建参考基因的 blast 数据库
    makeblastdb -in AP_scaffold.fasta -input_type fasta -dbtype nucl -title ap -out ap
# Building a new DB, current time: 04/23/2018 14:11:29
# New DB name:    /home/student/s21/exp3/ap
# New DB title:   ap
# Sequence type: Nucleotide
# Keep Linkouts: T
# Keep MBits: T
# Maximum file size: 1000000000B
# Adding sequences from FASTA; added 36 sequences in 0.943164 seconds.
```

3、已知蛋白序列的下载 根据基因组的物种来源，从 UniProt 数据库下载近缘物种（fungi）已知蛋白序列， 以 FASTA 格式保存。

Uniprot 数据库中检索 taxonomy:fungi AND reviewed:yes
下载所有序列，共 33161

4、同源基因搜索：

4.1、使用 tblastn 程序，把已知蛋白质序列和基因组草图序列建立的本地 BLAST 数据库进行比对，注意参数设置（如 e-value 设为 0.00001，建议输出格式 6 或 7。

```
#step3：tblastn
tblastn -db ap -query uniprot-taxonomy-reviewed.fasta -out ./tblastn_results.outfmt6
-evalue 1e-5 -outfmt 6 -num_threads 30
```

```
# tblastn -query uniprot-taxonomy_fungi.fasta -out seq7.blast -db AP_db -outfmt 7
-evalue 1e-5
tblastn -db ap -query uniprot-taxonomy-reviewed.fasta -out tblastn_results.outfmt7
-evalue 1e-5 -outfmt 7 -num_threads 30
```

4.2、编程处理 BLAST 比对结果文件，排除冗余项：

（1）不同物种的同一种蛋白在基因组上的匹配位置存在的重叠问题；

（2）同一物种的同一蛋白家族的不同成员蛋白在基因组上的匹配位置存在的重叠 问题；

（3）同一个蛋白在基因组上的不同位置的高相似区域问题——是家族成员问题， 还是冗余匹配；

算法思路：

1. 数据预处理：R 语言

1.1. 根据正链和互补链的配对区别分类：

例如：

	sub	start1	end1	start2	end2	score
sp|Q9P6J0|YHDC_SCHPO		14	99	4045	4359	54.3

此类序列为正链配对

	sub	start1	end1	start2	end2	score
sp|P38735|VMR1_YEAST		749	1102	949	17	126.0

此类序列为互补链配对

⇨

通过 R 语言整合后，正链配对的为 po_chain.txt，互补链配对的为 ne_chain.txt

⇨

以下步骤分别针对正链配对的和互补链配对需做两次处理，R 语言代码见附录。

1.2. 针对以上任意一文件，将属于同一个 scaffold（eg.scaffold_x）的不同蛋白，处理为同一个小组进行循环。

⇨ 正链配对的存在 17 个不同的 scaffold

k = Series([1,10,11, 12, 13, 14, 15, 16, 17, 2, 3, 4 ,5, 6, 7, 8, 9]) #pyhon 数组

⇨ 互补链配对的存在 19 个不同的 scaffold

k = Series([1,10,11,12,13,14,15,16,2,20,25,3,36,4,5,6,7,8,9]) #python 数组

2. 进一步数据处理：python 语言

注：分别针对正链配对、互补链配对以及不同的 scaffold 个数进行进一步处理，为了快速运行，以下阶段采用 python 语言进行数据处理，但是由于不够熟练，完整代码不能

2.1. 处理多外显子问题：#针对正链配对

```
from pandas import DataFrame,Series
import pandas as pd
mat = pd.read_table("po_chain.txt")

cmat=mat.sort_values(by="start2") #将每行蛋白在基因组上的起始位置进行排序
cmat['flag']=0 #增加一列”flag”为去冗余做准备
```

```
pro=pd.DataFrame()
for i in range(0,len(mat),1): #遍历同一个 scaffold 的每一行蛋白信息
  start_one = mat.iloc[i,3] #提取在基因组上的起始位置
  end_one = mat.iloc[i,4] #提取在基因组上的结束位置
  prot_region1 = end_one+1000 #设置 1000 为阈值

  for r in range(1,len(mat),1): #遍历 i 的下一行
    # 3=>start2; 4=>end2
    if
(mat.iloc[r,0]==mat.iloc[i,0])&(mat.iloc[r,3]>end_one)&(mat.iloc[r,3]<prot_regi
on1):
  #判断 i+1 行和 i 行名字是否相同，i+1 行信息在基因组上的起始位置是否在阈值的范围内
  #全部符合条件，则将整合好的信息添加在一个新的 dataframe 里，pro
  #不符合条件，则将第 i 行信息添加在 pro 里，遍历 i+1 行是否存在多外显子问题
      blast=mat[i:i+1]
      #name,start1,end1,start2,end2,score,flag
      blast.iat[1] = mat.iloc[i,1]
      blast.iat[2] = mat.iloc[r,2]
      blast.iat[3] = mat.iloc[i,3]
      blast.iat[4] = mat.iloc[r,4]
      blast.iat[5] = mat.iloc[i,5]+mat.iloc[r,5]
      pro.append(balst)
    else:
      pro.append(mat[i:i+1])
```

2.2. 冗余去除： #针对正链配对

#针对上一部分生成的名为”pro”的 dataframe，根据 overlap 进行去冗余，存在重叠的

```
for i in range(0,len(pro),1):
    for d in range(i+1,len(pro),1):
      if
(pro.iloc[i,0]==plo.iloc[d,0])&((pro.iloc[i,3]<pro.iloc[d,3])&(pro.iloc[i,4]>pro.iloc[d,3]
))|((pro.iloc[i,3]>pro.iloc[d,3])&(pro.iloc[i,4]<pro.iloc[d,4]))|((pro.iloc[i,3]<pro.iloc[d
,4])&(pro.iloc[i,4]>pro.iloc[d,4])):
      #根据在基因组上的对应问题，去除冗余
        pro[d,5]=1
      else:
        pro[d,5]=0 ##”flag”列
#如果符合条件，给 pro 该行信息”flag”列设置为 1
#如果不符合条件，pro 的”flag”列为原始值 0
```

2.3. 整合 pro 信息，把 flag 为 0 的删除
2.4. 循环 k 的 Series，遍历所有 scaffold
2.5. 重复执行配对链的

5、从头预测:

5.1、从网上搜索、下载并安装基因预测相关软件【至少 1 个】;

```
#测试平台：Win10 下的 VMware Workstation Pro 虚拟机——操作系统 Ubuntu14.04
#下载适合自己系统的版本（http://exon.gatech.edu/GeneMark/license_download.cgi）
#这里下载测试的是 GeneMark-ES / ET v.4.33 （Linux 64 位）版本
#解压缩之后程序在 home 目录下
cd /home/gm_et_linux_64/gmes_petap
cp gm_key ~/.gm_key
#genemark 使用 perl 语言编写的，需要系统安装 perl
#查看 perl 的安装目录 which perl #一般安装在/usr/bin/、或/usr/bin/perl 目录下
#genemark 的正常使用还需要安装一些 perl 扩展包
perl -MCPAN -e shell
    =》yes =》sudo =》yes =》也可以全默认选择
    install YAML
    install Hash::Merge
    install Logger::Simple
    install Parallel::ForkManager
    install Getopt::Long
    install File::Spec
    install File::Path
    install Data::Dumper
    =》exit
```

5.2、使用该软件对基因组序列进行基因预测分析，保存预测基因编码的多肽

----nuc_seq.fna➔nuc.fasta

```
#使用下载的 AP_genome_sequencing
./gmes_petap.pl --prediction --predict_with ./heu_dir/heu_05_gcode_1_gc_50.mod
--sequence AP_scaffold.fasta
#根据结果文档（gtf），从基因组草图中提取序列
./get_sequence_from_GTF.pl ./AP_genome_draft/genemark.gtf AP_scaffold.fasta
 #prot_seq.faa 是蛋白质序列----》5.3 步操作
  #nuc_seq.fna 是核酸序列----》5.2 步操作
```

5.3、使用 blastp 程序对该预测基因与已知蛋白序列进行比对，以此来鉴别预测的基因。

---prot_seq.fna➔pro.fasta

```
  #使用小型机 账号 s21
  cd exp3
  makeblastdb -in uniprot-taxonomy-reviewed.fasta -input_type fasta -title
uniprot_fungi-reviewed -dbtype prot -out uniprot_fungi
```

```
 #
# Building a new DB, current time: 04/27/2018 18:54:59
# New DB name:      /home/student/s21/exp3/uniprot_fungi
# New DB title:    uniprot_fungi-reviewed
# Sequence type: Protein
# Keep Linkouts: T
# Keep MBits: T
# Maximum file size: 1000000000B
# Adding sequences from FASTA; added 33237 sequences in 2.49843 seconds.
blastp -db uniprot_fungi -query pro.fasta -out ./blastp_results.outfmt6 -evalue 1e-5
-outfmt 6 -max_target_seqs 1 -num_threads 30
```

5.4、整合 blastp 结果和 genemark.gtf，标记比对结果的蛋白名称，整合为文件”merge.txt”。

```
#R 语言
#### step7 add infor
setwd("E:/2018.3-2018.7----大三下/基因组信息学/第三次作业中间文件/7 从头预测")
c <- read.table("genemark.gtf",head=F,sep="\t")
                # V1                V2              V3      V4      V5 V6 V7 V8
V9
# 1 scaffold_11 GeneMark.hmm              exon    4535    4644    0    -    . gene_id 1_g;
transcript_id 1_t;
# 2 scaffold_11 GeneMark.hmm    stop_codon    4535    4537    .    -    0 gene_id 1_g;
transcript_id 1_t;
select <- unlist(strsplit(as.character(c$"V9")," "))
del <- seq(1, length(select), by = 2)
scaffold2<- select[-del]
select2 <- unlist(strsplit(as.character(c$"V9")," "))
sc <- unlist(strsplit(scaffold2,";"))
del <- seq(2, length(select2), by = 2)
gene <-sc[-del]
gene<-as.data.frame(gene)
new <- cbind(c,gene)

pr <- read.table("new//blastp_results.outfmt6")
pr <- pr[,1:2]
colnames(pr) <- c("gene","pr_name")


cc<-merge(new,pr,by="gene",all.x=T)
```

三．讨论分析：

1.对于蛋白在基因组上的匹配问题，需要主要双链匹配时是否存在正链和负链的匹配问题，同时要分辨出冗余信息以及家族蛋白的问题。针对本部分的问题，由于不够熟练应用 python 语言，所以完整程序不能附上。同时在使用算法分析的同时要注意时间复杂度不可过大。

2.使用本地数据进行比对时，可以选择匹配结果的个数（参数：-max_target_seqs），该参数需要与否需针对不同问题，如果需要查看所有的匹配情况则可以不设置，如果针对蛋白的鉴别，则无需查看所有的匹配情况，可以设置目标序列个数为 1

四．附录

4.2.去冗余问题前的预处理数据：

```
###R 语言去除冗余
setwd("E:/2018.3-2018.7----大三下/基因组信息学/第三次作业中间文件")
outfmt6 <- read.table("outfmt6.txt",sep="\t",head=F)
colnames(outfmt6)<-c("sub","tar","X1","X2","X3","X4","start1","end1","start2","end2","p","score
")
rownames(outfmt6)<-paste("X",1:length(outfmt6[,1]))
##区分正链匹配和互补链匹配
start1<-as.data.frame(outfmt6$"start1")
end1<-as.data.frame(outfmt6$"end1")
dif <- end1-start1
length(dif[dif<0])
## 0
##说明全部有小到大排列
start2<-as.data.frame(outfmt6$"start2")
end2<-as.data.frame(outfmt6$"end2")
dif2 <- end2-start2
rownames(dif2) <- rownames(outfmt6)
length(dif2[dif2[1]<0])
##[1] 103592
##近一半的互补链匹配
tf<-dif2<0

norp<-c()     ##正链配对
comp<-c()     ##互补链配对

for(i in 1:length(outfmt6[,1])){
      if(tf[i,]==TRUE){
            comp <- rbind(comp,outfmt6[i,])
      }
      if(tf[i,]==FALSE){
            norp <- rbind(norp,outfmt6[i,])
      }
}
dim(norp) #89354       12
dim(comp) #103592         12
#  使用 unbuntu 生成
# setwd("E:/2018.3-2018.7----大三下/基因组信息学/第三次作业中间文件")
```

```
# norp <- read.table("com_pair.txt",head=T,sep="\t")       ##正链配对
# comp <- read.table("nor_pair.txt",head=T,sep="\t")       ##负链配对


mat <- comp
mat <- mat[,-c(3,4,5,6,11)]
mat_order <- mat[order(mat[,2]),]
scaffold <- unlist(strsplit(as.character(mat_order[,2]),"_"))
del <- seq(0, length(scaffold), by = 2)
scaffold <- scaffold[del]
mat_deal1 <- cbind(mat_order,scaffold)
mat_deal1 <- mat_deal1[,-2]
length(levels(mat_deal1$"scaffold"))

write.table(mat_deal1,"ne_chain.txt",sep="\t")

mat <- norp
mat <- mat[,-c(3,4,5,6,11)]
mat_order <- mat[order(mat[,2]),]
scaffold <- unlist(strsplit(as.character(mat_order[,2]),"_"))
del <- seq(0, length(scaffold), by = 2)
scaffold <- scaffold[del]
mat_deal1 <- cbind(mat_order,scaffold)
mat_deal1 <- mat_deal1[,-2]
length(levels(mat_deal1$"scaffold"))

write.table(mat_deal1,"po_chain.txt",sep="\t")
```

```
###################### edition1 ##################################################
#! /usr/bin/env python3
#! coding:utf-8
# author:Feiyang Zhao
import pandas as pd

#remove duplicated blast results which are in positive chains
def rm_duplicated_pos(filename):
    #open the target-file
    f = open(filename,'r')
    #get the data
    data = f.readlines()
    f.close()
    #split every line by '\t' in order to use their indexs
    temp = []
    for x in data:
        temp.append(x.split('\t'))

    #get the unique protien names
    protein = []
    for x in temp:
        protein.append(x[0][3:9])
    protein = list(set(protein))

    #classify the target-data by different protein names
    firstdata = []
    for x in protein:
        t = []
        for y in temp:
            if y[0][3:9] == x:
                t.append(y)
        firstdata.append(t)

    #remove the duplicated part
    finaldata = []
    #go throuth every group classified by protein name
    for x in firstdata:
        #this protein has only one blast result, so remain it
        if len(x) == 1:
            finaldata.append(x)
        else:
            for i in range(len(x)):
                #get the aim1's startsite and endsite on nuclein
                (nucstart, nucend) = x[i][8:10]
```

```
                    for j in range(i+1,len(x)):
                        #get the aim2's startsite and endsite on nuclein
                        (nucstart1, nucend1) = x[j][8:10]
                        '''there are three situations we must consider, first is the aim1 cotaining
aim2
                        second is they having left overlap and the last is they having right
overlap,
                        but the first situation can be included in the other situations, so I set the
                        cut-off length of overlap is 10000 with comparing their scores(remain
the high
                        score result), but I found a problem that a child set may has better socre
                        than its parent set'''
                        if (int(nucend1) - int(nucstart) > 10000 or int(nucend) - int(nucstart1) >
10000) \
                        and (int(nucend) - int(nucstart)) < (int(nucend1) - int(nucstart)):
                        #and float(x[i][11].replace('\n','')) < float(x[j][11].replace('\n','')):
                            break
                    #if this happened, it means the aim1 is one of our dream results
                    if j == len(x):
                        finaldata.append(x[i])
    for x in finaldata:
        x[0][11] = x[0][11].replace('\n','')
        pd.DataFrame(x).to_csv(filename+'rmlen',sep='\t',mode='a',header=False,index=False)

#remove duplicated blast results which are in negative chains
def rm_duplicated_neg(filename):
    f = open(filename,'r')
    data = f.readlines()
    f.close()
    temp = []
    for x in data:
        temp.append(x.split('\t'))

    protein = []
    for x in temp:
        protein.append(x[0][3:9])
    protein = list(set(protein))

    firstdata = []
    for x in protein:
        t = []
        for y in temp:
            if y[0][3:9] == x:
                t.append(y)
```

```
            firstdata.append(t)

        finaldata = []
        for x in firstdata:
            if len(x) == 1:
                finaldata.append(x)
            else:
                for i in range(len(x)):
                    (nucend, nucstart) = x[i][8:10]
                    for j in range(i+1,len(x)):
                        (nucend1, nucstart1) = x[j][8:10]
                        if (int(nucend1) - int(nucstart) > 10000 or int(nucend) - int(nucstart1) >
10000) \
                        and (int(nucend) - int(nucstart)) < (int(nucend1) - int(nucstart)):
                        #and float(x[i][11].replace('\n','')) < float(x[j][11].replace('\n','')):
                            break
                    if j == len(x):
                        finaldata.append(x[i])
        for x in finaldata:
            x[0][11] = x[0][11].replace('\n','')
            pd.DataFrame(x).to_csv(filename+'rmlen',sep='\t',mode='a',header=False,index=False)

#give names to the original data
columns = ['A','B','C','D','E','F','G','H','I','J','K','L']
#read the original data
initdata = pd.read_table('res',sep='\t',header=None,names=columns)
#get the positive data
positive = initdata[initdata.I - initdata.J < 0].to_csv('positive',sep='\t',header=False,index=False)
#get the negative data
negative = initdata[initdata.I - initdata.J > 0].to_csv('negative',sep='\t',header=False,index=False)
rm_duplicated_pos('positive')
rm_duplicated_neg('negative')
#read the positive data which has removed duplicated results
pos = pd.read_table('positiverm',sep='\t',header=None,names=columns)
#read the negative data which has removed duplicated results
neg = pd.read_table('negativerm',sep='\t',header=None,names=columns)
#merge the positive and negative datas
result = pd.concat([pos,neg],axis=0)
#sort the data by scores
result = result.sort_values(by='L',ascending=False)
#output the result
result.to_csv('rmresult',sep='\t',header=False,index=False)
```

```
########################## edition2 ################################################
#! /usr/bin/env python3
#! coding:utf-8
# author:Feiyang Zhao
import pandas as pd

#remove duplicated blast results which are in positive chains
def rm_duplicated_pos(filename):
    #open the target-file
    f = open(filename,'r')
    #get the data
    data = f.readlines()
    f.close()
    #split every line by '\t' in order to use their indexs
    temp = []
    for x in data:
        temp.append(x.split('\t'))

    #get the unique protien names
    protein = []
    for x in temp:
        protein.append(x[0][3:9])
    protein = list(set(protein))

    #classify the target-data by different protein names
    firstdata = []
    for x in protein:
        t = []
        for y in temp:
            if y[0][3:9] == x:
                t.append(y)
        firstdata.append(t)

    #remove the duplicated part
    finaldata = []
    #go throuth every group classified by protein name
    for x in firstdata:
        #this protein has only one blast result, so remain it
        if len(x) == 1:
            finaldata.append(x)
        else:
            for i in range(len(x)):
                #get the aim1's startsite and endsite on nuclein
                (nucstart, nucend) = x[i][8:10]
```

```
                    for j in range(i+1,len(x)):
                        #get the aim2's startsite and endsite on nuclein
                        (nucstart1, nucend1) = x[j][8:10]
                        '''there are three situations we must consider, first is the aim1 cotaining
aim2
                        second is they having left overlap and the last is they having right
overlap,
                        but the first situation can be included in the other situations, so I set the
                        cut-off length of overlap is 10000 with comparing their scores(remain
the high
                        score result), but I found a problem that a child set may has better socre
                        than its parent set'''
                        if (int(nucend1) - int(nucstart) > 10000 or int(nucend) - int(nucstart1) >
10000) \
                        and (int(nucend) - int(nucstart)) < (int(nucend1) - int(nucstart)):
                        #and float(x[i][11].replace('\n','')) < float(x[j][11].replace('\n','')):
                            break
                    #if this happened, it means the aim1 is one of our dream results
                    if j == len(x):
                        finaldata.append(x[i])
    for x in finaldata:
        x[0][11] = x[0][11].replace('\n','')
        pd.DataFrame(x).to_csv(filename+'rmlen',sep='\t',mode='a',header=False,index=False)

#remove duplicated blast results which are in negative chains
def rm_duplicated_neg(filename):
    f = open(filename,'r')
    data = f.readlines()
    f.close()
    temp = []
    for x in data:
        temp.append(x.split('\t'))

    protein = []
    for x in temp:
        protein.append(x[0][3:9])
    protein = list(set(protein))

    firstdata = []
    for x in protein:
        t = []
        for y in temp:
            if y[0][3:9] == x:
                t.append(y)
```

```
            firstdata.append(t)

        finaldata = []
        for x in firstdata:
            if len(x) == 1:
                finaldata.append(x)
            else:
                for i in range(len(x)):
                    (nucend, nucstart) = x[i][8:10]
                    for j in range(i+1,len(x)):
                        (nucend1, nucstart1) = x[j][8:10]
                        if (int(nucend1) - int(nucstart) > 10000 or int(nucend) - int(nucstart1) >
10000) \
                        and (int(nucend) - int(nucstart)) < (int(nucend1) - int(nucstart)):
                        #and float(x[i][11].replace('\n','')) < float(x[j][11].replace('\n','')):
                            break
                    if j == len(x):
                        finaldata.append(x[i])
        for x in finaldata:
            x[0][11] = x[0][11].replace('\n','')
            pd.DataFrame(x).to_csv(filename+'rmlen',sep='\t',mode='a',header=False,index=False)

#give names to the original data
columns = ['A','B','C','D','E','F','G','H','I','J','K','L']
#read the original data
initdata = pd.read_table('res',sep='\t',header=None,names=columns)
#get the positive data
positive = initdata[initdata.I - initdata.J < 0].to_csv('positive',sep='\t',header=False,index=False)
#get the negative data
negative = initdata[initdata.I - initdata.J > 0].to_csv('negative',sep='\t',header=False,index=False)
rm_duplicated_pos('positive')
rm_duplicated_neg('negative')
#read the positive data which has removed duplicated results
pos = pd.read_table('positiverm',sep='\t',header=None,names=columns)
#read the negative data which has removed duplicated results
neg = pd.read_table('negativerm',sep='\t',header=None,names=columns)
#merge the positive and negative datas
result = pd.concat([pos,neg],axis=0)
#sort the data by scores
result = result.sort_values(by='L',ascending=False)
#output the result
result.to_csv('rmresult',sep='\t',header=False,index=False)
```