Code that I created or heavily modified from Dr. Balaji or Alvaro is highlighted in yellow.

HW3 Library

```
##### Hw 3 Library ####################

library(gcmr)
library(dplyr)
library(verification)
library(HiddenMarkov)
library(MASS)
library(ggplot2)
library(ggthemes)
library(data.table)
library(leaps)        # to provide combinations
library(MPV)          # for calculating PRESS

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW3/data")

# read all data
Q_mon=read.table('LeesFerry-monflows-1906-2006.txt', header=F)
colnames(Q_mon)=c('Year','Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec')
Q_mon[,-1]=Q_mon[,-1]/(10^6)


Apr1Forecast=read.table('Apr1Forecast.txt', sep=',',header=T)
Feb1Forecast=read.table('Feb1Forecast.txt', sep=',',header=T)
Jan1Forecast=read.table('Jan1Forecast.txt', sep=',',header=T)


# make Forecast data and monthly flows have matching years

a=range(Q_mon$Year)
b=range(Apr1Forecast$wyears)
start=max(a[1],b[1])
end=min(a[2], b[2])

Q_mon=Q_mon[Q_mon$Year %in% start:end,]
spring=which(colnames(Q_mon) %in% c('Apr', 'May', 'Jun'))
Q_spring=Q_mon[,spring]


features=list(Jan=Jan1Forecast, Feb=Feb1Forecast, Apr=Apr1Forecast)

for (i in 1:length(features)){

  features[[i]]=features[[i]][features[[i]]$wyears %in% start:end,]
  # features[[i]]=data.frame(Q_spring=apply(Q_mon[,5:7],1,sum), features[[i]])
}
```

```
rm(a, b, start, end, spring,i)

############################### Read Winter SST Anomaly data
#########################

readSST=function(){
  ### Lat - Long grid..
  # information obtained from:
http://iridl.ldeo.columbia.edu/SOURCES/.KAPLAN/.EXTENDED/.v2/

  ygrid=seq(-87.5,87.5,by=5)
  ny=length(ygrid) # number of latitudinal locations

  xgrid=seq(27.5,382.5,by=5)
  nx=length(xgrid) # number of longitudinal locations

  nglobe = nx*ny
  # creation of spatial grid
  xygrid=matrix(0,nrow=nx*ny,ncol=2)
  i=0
  for(iy in 1:ny){
    for(ix in 1:nx){
      i=i+1
      xygrid[i,1]=ygrid[iy]
      xygrid[i,2]=xgrid[ix]
    }
  }


  ################### Read Gridded monthly global sea surface temp (SST) anomalies (AKA
Kaplan SST)
  #

  nyrs = 118    #Nov-Mar 1901 - Nov-Mar 2018
  data=readBin(con='http://civil.colorado.edu/~balajir/CVEN6833/HWs/HW-2/Kaplan-SST-
DJFM1900-DJFM2018.r4', what='numeric', n=(nx*ny*nyrs),
               size=4, endian = 'swap')

  data = array(data = data, dim=c( nx, ny, nyrs ) )
  data1=data[,,1]


  # the lat -long data grid..

  index=1:(nx*ny)

  index1=index[data1 < 20 & data1 != "NaN"]   # only non-missing data.
  # what is less than 20 for?
  xygrid1=xygrid[index1,] # these are the locations of remaining data, corresponding to the cols
sstannavg
```

```
  nsites=length(index1)
  data2=data1[index1]

  ### SSTdata matrix - rows are seasonal (i.e. one value per year)
  ## and columns are locations
  sstannavg=matrix(NA,nrow=nyrs, ncol=nsites)

  for(i in 1:nyrs){
    data1=data[,,i]
    index1=index[data1 < 20 & data1 != "NaN"]
    data2=data1[index1]
    sstannavg[i,]=data2
  }

  indexgrid = index1
  rm("data")  #remove the object data to clear up space

  lon=sort(unique(xygrid[,2]))
  lat = sort(unique(xygrid[,1]))

  return(list(sst=sstannavg, nglobe=nglobe, indexgrid=indexgrid, lon=lon,lat=lat))

}



######################################## PCA function with scree plot
#################################

# all code from Alvaro, Nathan just turned it into a function

pca=function(data, scale=T){

  if(scale==T){
    scaled_data=scale(data)
  } else {
    scaled_data=data
  }

  covariance_mat=var(scaled_data) # covariance matrix
  zsvd=svd(covariance_mat) # SVD results in three matrices. u, eigen value diagonal (lamdas),
V. u is the matrix of loading scores, AKA phi, AKA eigen vector matrix
  pcs=scaled_data %*% zsvd$u # Z = X times u matrix
  lambdas = zsvd$d/sum(zsvd$d) # d is variance in each PC. Lambda is proportion of variance
explained

  scree=ggplot() +
    geom_line(mapping = aes(c(1:25), lambdas[1:25])) +
    geom_point(mapping = aes(c(1:25), lambdas[1:25]), shape = 21, size = 3, color = "gray30",
fill ="cadetblue1") +
    labs(title = "Eigen Spectrum",x = "Modes",y = "Frac. Var. explained")+
```

```
    theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

  pca_results=list()
  pca_results[['pcs']]=pcs
  pca_results[['u']]=zsvd$u
  pca_results[['PVE']]=lambdas
  pca_results[['scree_plot']]=scree


  return(pca_results)
}


########################################## KNN and bootstrapping
##########################################

knn = function(df.train, preds, remove_cols, bootstrap=TRUE, scale_dist=T){

  #remove year and flow from predictor input so just covariates remain
  preds = preds[,-remove_cols]

  #choose k to either be the length of full dataset (useful if bootstrapping later) or the optimal
sqrt(n) (use if not bootstrapping)
  #could also choose some other user defined value for k

  if (bootstrap == T ){
    k = nrow(df.train)
  } else {
    k = ceiling(sqrt(nrow(df.train)))
  }


  #number of predictors
  np = ncol(preds)

  #initialize distance matrix
  dist.mat = matrix(NA, nrow = nrow(df.train), ncol = np)

  #calc distance for each neighbor (loop thru each predictor)

  for (r in 1:nrow(dist.mat)){

    dist.mat[r,]=unlist(preds[1,]-df.train[r,-remove_cols])

  }

    # i = 1
    # for(i in 1:np){
    #   dist.mat[,i] = preds[1,i] - df.train[,i]
    # }
```

```
  if (scale_dist==T){
    dist.mat=scale(dist.mat)
  }

    dist.mat=dist.mat^2 #square distance


  #calc Euclidean distance for each row (year) by summing distances of each predictor
  df.train$dist = sqrt(apply(dist.mat, 1, sum))

  #select the k smallest distances to return k nearest neighbors
  neighbs = slice_min(df.train, order_by = dist, n = k)

  return(neighbs)

}


neighbor_bootstrapping = function(neighbs, nsim, predict_col){

  #calc inverse distance weights for k-neighbors
  k = nrow(neighbs)
  knn.wts = (1/(1:k))/sum(1/(1:k))

  #bootstrap simulated flows from neighbors based on IDW scheme
  sims = sample(neighbs[,predict_col], nsim, replace = T, prob = knn.wts)

  return(sims)
}


############################ Hidden Markov Model ##############################


# function from Dr. Balaji
get.named.parlist <- function(x,m,dist,ic,...){
  require(MASS)
  fit <- fitdistr(x,dist,...)
  np <- length(fit$estimate)
  pars <- vector('list',np)
  names(pars) <- names(fit$estimate)

  init <- lapply(fit$estimate,max)
  names(init) <- names(fit$estimate)


  for(j in 1:m){
    #print(j)
    #browser()

    #browser()
```

```
    this.fit <- fitdistr(x[ntile.ts(x,m) == (j-1)],dist,init,...)
    #for(k in 1:np)
    #    pars[[k]][j] <- this.fit$estimate[k]
    for(k in 1:np)
      pars[[k]][j] <- fit$estimate[k]
    if(dist == 'normal'){
      if(ic == 'same.both'){
        pars[[k]][j] <- mean(x)
        pars[[k]][j] <- sd(x)
      } else if( ic == 'same.sd'){
        pars[[k]][j] <- mean(x[ntile.ts(x,m) == (j-1)])
        pars[[k]][j] <- sd(x)
      }else{
        pars[[k]][j] <- mean(x[ntile.ts(x,m) == (j-1)])
        pars[[k]][j] <- sd(x[ntile.ts(x,m) == (j-1)])
      }
    }
  }
  pars
}

# from Dr. Balaji
Pi_init <- function(n,type='uniform'){
  matrix(rep(1/n,n^2),n)
}
delta_init <- function(n, type='uniform'){
  d <- rnorm(n)^2
  d/sum(d)
}

# from Dr. Balaji
ntile.ts <-
  function(x, n, limit.type = 'prob', tie = 1, altobs = NULL ){
    # returns an integer vector corresponding to n states broken by equal
    # probability or equal distance
    #
    limit <-
      if(limit.type == 'prob')
        quantile(x,seq(0,1,1/n))
    else if(limit.type == 'equal')
      seq(min(x),max(x),by=diff(range(x))/n)

    if(!is.null(altobs)) limit <- quantile(altobs,seq(0,1,1/n))

    b <- integer(length(x))

    for(i in 1:(n+1)){
      filter <-
        if(tie == 1)
          x >= limit[i] & x <= limit[i+1]
      else
```

```
      x > limit[i] & x <= limit[i+1]

    #only need to set the 1's because b is already 0's
    b[filter] <- as.integer(i-1)
  }

  if(class(x) == 'ts')
    return(ts(b,start=start(x),end=end(x)))
  else
    return(b)
}

# AIC (or BIC) for HMM

AIC_hmm=function(hmm, k=2){
 LL=hmm$LL
 n=length(hmm$x)

 n_pdf=length(hmm$pm) # number of parameters to fit one marginal distribution
 n_states=nrow(hmm$Pi) # number of states
 n_transition=nrow(hmm$Pi)*ncol(hmm$Pi) # number of transition probabilities
 npar=n_states*n_pdf+n_transition

 return(-2*LL+k*npar)
}

ggplot_stationary_hmm <- function(x,binwidth=NULL,res=1000,cols=NULL,...){

 m <- length(x$delta)
 dens <- matrix(0,nrow=m+1,ncol=res)
 r <- extendrange(x$x,f=.05)
 xrange <- seq(r[1],r[2],len=res)
 delta <- statdist(x$Pi)
 if(is.null(binwidth)) binwidth <- diff(range(x$x))/8
 for(i in 1:m){

  if(x$distn == 'gamma'){
    dens[i,] <- delta[i]*dgamma(xrange,shape=x$pm$shape[i],rate=x$pm$rate[i])
  }else if(x$distn == 'norm'){
    dens[i,] <- delta[i]*dnorm(xrange,mean=x$pm$mean[i],sd=x$pm$sd[i])
  }else{
    stop('Distribution not supported')
  }

  dens[m+1,] <- dens[m+1,] + dens[i,]
 }

 p <- ggplot()+
  geom_histogram(data=data.frame(x=as.vector(x$x)),aes(x=x,y=..density..),
                 binwidth=binwidth,fill='white',color='black')+
  theme_bw()
```

```
  dt <- data.table(x=numeric(0),y=numeric(0), state=integer(0))
  for(i in 1:m)
    dt <- rbind(dt, data.table(x=xrange,y=dens[i,], state=i))
  dt$state <- factor(dt$state)

  p <- p + geom_line(data=dt,aes(x=x,y=y,color=state)) +
    geom_line(data=data.frame(x=xrange,y=dens[m+1,]),aes(x=x,y=y),color='black',size=1) +
    scale_color_tableau() +
    scale_x_continuous(limits=r)
  p

}

statdist <- function(tpm){

  m <- nrow(tpm)
  ones <- rbind(rep(1,m))
  I <- diag(rep(1,m))
  U <- matrix(rep(1,m^2),m)
  as.vector(ones %*% solve(I - tpm + U))

}


#################### GLM ###############

### Find best GLM
GLM_fit = function(Pm, X, family, month) {

  if (family == "gamma") {
    links = c("log", "inverse","identity")

    # clean data and remove zeros
    Pm = ifelse(Pm <=0, runif(1, 0.0001, 0.001), Pm)

  } else if (family == "binomial"){
    links = c("logit", "probit", "cauchit")
  } else if (family == "gaussian"){
    links = c("identity")
  }

  N = length(Pm)

  combs = leaps(X,Pm, nbest=25)     #  GEt upto 25 combinations for each
  # number of predictors
  combos = combs$which
  ncombos = length(combos[,1])
  glm_press=vector(length = length(links))
  best_comb=vector(length = length(links))
  xpress=1:ncombos
```

```
xmse = 1:ncombos

for(j in 1:length(links)) {
  aux_var=1 # allow to fit glm
  ##remove small values for gamma distribution
  if (family == "gamma"){
    if (links[j]=="identity"){
      if( month==1){
        Pm[Pm<=0.005]=0.005
      }else{
        Pm[Pm<=2.5]=2.5
      }

    }else if (links[j]=="inverse" & month==1){
      Pm[Pm<=25]=25 # it is necessary to modific significantly the small values,
      #so, "inverse" is not fitted for January
      aux_var=0
    }
  }
  if (aux_var==1)
  {for(i in 1:ncombos) {
    xx = X[,combos[i,]]
    xx=as.data.frame(xx)
    if (family == "gamma"){
      zz=glm(Pm ~ ., data=xx, family = Gamma(link=links[j]), maxit=500)
    }else if (family == "binomial"){
      zz=glm(Pm ~ ., data=xx, family = binomial(link=links[j]), maxit=500)
    }else if (family == "gaussian"){
      zz=glm(Pm ~ ., data=xx, family = gaussian(link=links[j]), maxit=500)
    }
    xpress[i]=PRESS(zz)
    xmse[i] = sum((zz$res)^2) / (N - length(zz$coef))
    #print(xpress[i])
  }}
  if (aux_var==1){
    # Test using PRESS objective function
    glm_press[j]=min(xpress)
    best_comb[j]=which.min(xpress)
  }else{
    # Test using PRESS objective function
    glm_press[j]=200000
    best_comb[j]=which.min(xpress)
  }
}

press_df = data.frame(glm_press)
rownames(press_df) = links[1:length(links)]

print("Results of PRESS for bestfit GLM")
print(press_df)
print(best_comb[which.min(glm_press)])
```

```
  sprintf("Choosing the GLM which minimizes PRESS: %s family and %s link function.", family,
links[which.min(glm_press)])
  xx = X[,combos[best_comb[which.min(glm_press)],]]
  xx=as.data.frame(xx)
  if (family == "gamma") {
    bestmod = glm(Pm ~ ., data = xx, family = Gamma(link=links[which.min(glm_press)]))
  } else if (family == "binomial") {
    bestmod = glm(Pm ~ ., data = xx, family = binomial(link=links[which.min(glm_press)]))
  }  else if (family == "gaussian") {
    bestmod = glm(Pm ~ ., data = xx, family = gaussian(link=links[which.min(glm_press)]))
  } else {
    print("Error!")
  }
  bestmod$Call$LINK=links[which.min(glm_press)]
  bestmod$Call$PRESS=PRESS(bestmod)
  bestmod$Call$combo=combos[best_comb[which.min(glm_press)],]
  return(bestmod)
}



#################### climatology forecasts of Q ########################

q_maf=features$Jan$q_maf
Qterc=quantile(q_maf, c(0.333, 0.6667))
historical_climate=rep(NA, length(q_maf))

for (i in 1:length(q_maf)){
  if ( q_maf[i] < Qterc[1]){

    historical_climate[i]=1

  } else if ( q_maf[i] >= Qterc[1] & q_maf[i] < Qterc[2] ) {

    historical_climate[i]=2


  } else {

    historical_climate[i]=3

  }


}

climatology=c(1/3, 1/3, 1/3)
```

P1: KNN

```
# CVEN 5353
# HW3
# P1
# Nathan Bonham

rm(list=ls())
setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW3")
source('HW3 Library.R')


############################## KNN-Conditional Ensemble Forecast/Simulation
####################


nsim=150
predict_col=which(colnames(features$Jan)=='q_maf')
#remove_cols=which(colnames(features$Jan) %in% c('Q_spring', 'wyears') ) # Use all features
remove_cols=which(!(colnames(features$Jan) %in% c('amo', 'pdo', 'enso', 'tmax_C',
'win.pcp_in')) ) # keep these only

q_mean=matrix(NA, ncol=length(features), nrow=nrow(features$Jan))
q_median=matrix(NA, ncol=length(features), nrow=nrow(features$Jan))

terc_mat=matrix(NA, nrow=nrow(features$Jan), ncol=3)
terc_forecast=list(Jan=terc_mat, Feb=terc_mat, Apr=terc_mat)

for (i in 1: length(features)){ # loop through Jan, Feb, Apr lead times

  df=features[[i]]

  for (t in 1:nrow(df)){ # loop through years

    df.train=df[-t,] # remove current year for training data
    preds=df[t,] # store current year to predict

    neighbs=knn(df.train, preds, remove_cols = remove_cols, bootstrap = TRUE, scale_dist = F)

    sims=neighbor_bootstrapping(neighbs = neighbs, nsim=nsim, predict_col = predict_col)

    p1=sum(sims < Qterc[1])/length(sims)
    p3=sum(sims > Qterc[2])/length(sims)
    p2=1-(p3+p1)
    p=c(p1,p2,p3)

    terc_forecast[[i]][t,]=p
    q_mean[t, i]=mean(sims)
    q_median[t,i]=median(sims)

  }
```

```
  print(i)
}

################################ Plot ##########################

r=3
c=2

par(mfrow=c(r,c), mar=c(5,5,2,2))

Q=features$Jan$q_maf

xlab=c('Jan Forecast [MAF]', 'Feb Forecast [MAF]', 'Apr Forecast [MAF]')

for (i in 1:length(features)){
  yhat=q_mean[,i]
  plot(x=yhat, y=Q, xlab=xlab[i], ylab='Observed [MAF]', main='')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((yhat-Q)^2))
  rpss=rps(obs=historical_climate, pred=terc_forecast[[i]], baseline=climatology)$rpss
  corr=cor(Q, yhat)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep=''),
                                 paste('rpss=', round(rpss,2), sep='')), cex=1.1)

  if (i ==1){
   mtext('Mean of KNN ensemble', line=.3)
  }

  yhat=q_median[,i]
  plot(x=yhat, y=Q, xlab=xlab[i], ylab='Observed [MAF]', main='')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((yhat-Q)^2))
  rpss=rps(obs=historical_climate, pred=terc_forecast[[i]], baseline=climatology)$rpss
  corr=cor(Q, yhat)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep=''),
                                 paste('rpss=', round(rpss,2), sep='')), cex=1.1)

  if (i ==1){
    mtext('Median of KNN ensemble', line=.3)
  }



}
```

P2: Copula Regression

```
# CVEN 6833
# HW3
# P2 Conditional Forecast/Simulation - Copulas

rm(list=ls())

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW3")

source('HW3 Library.R')

############################################### Copulas
##########################################

# Using copula regression, simulate/forecast Spring streamflow on the
# Colorado River at Lees Ferry based on suite of predictors, also called, 'feature vectors', at
# three lead times - Jan 1, Feb 1 and Apr 1.

# (i) use gcmr package. There will not be an ensemble forecast, but a mean value

#  fit model to each month independently


Apr=Q_mon$Apr
May=Q_mon$May
Jun=Q_mon$Jun

par(mfrow=c(1,3))

hist(Apr, xlab='Flow [MAF]', main='April') # choose Gamma.marg
hist(May, xlab='Flow [MAF]', main='May')
hist(Jun, xlab='Flow [MAF]', main='June')

list_names=names(features)
forecast=list()

set.seed(5)

# find index of features you want to keep

keep_index=which(colnames(features$Jan) %in% c('amo', 'pdo', 'enso', 'tmax_C', 'win.pcp_in'))

for (i in 1:length(features)){ # loop through features for different lead times

  df=data.frame(Apr, features[[i]][,keep_index])
  Amod=gcmr(Apr~., data=df,marginal = Gamma.marg, cormat=ind.cormat, model=T)


  df=data.frame(May, features[[i]][,keep_index])
  Mmod=gcmr(May~., data=df,marginal = Gamma.marg, cormat=ind.cormat, model=T)
```

```
  df=data.frame(Jun, features[[i]][, keep_index])
  Jmod=gcmr(Jun~., data=df,marginal = Gamma.marg, cormat=ind.cormat, model=T)

  yhat.df=data.frame(Apr=Amod$fitted.values, May=Mmod$fitted.values,
Jun=Jmod$fitted.values)

  forecast[[list_names[i]]]=yhat.df # store forest in list for each lead time

}


########## one to one plot ############################

r=3
c=3

par(mfrow=c(r,c), mar=c(5,5,2,2))

for (i in 1:length(forecast)){
  df=forecast[[i]]
  plot(x=df$Apr, y=Apr, xlab='Forecast', ylab='Observed', main='April')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((df$Apr-Apr)^2))
  corr=cor(Apr, df$Apr)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep='')), cex=1.1)


  plot(x=df$May, y=May,xlab='Forecast', ylab='Observed', main='May')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((df$May-May)^2))
  corr=cor(May, df$May)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep='')), cex=1.1)

  plot(x=df$Jun, y=Jun,xlab='Forecast', ylab='Observed', main='June')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((df$Jun-Jun)^2))
  corr=cor(Jun, df$Jun)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep='')), cex=1.1)
}

################ Alternatively, fit a model the entire spring time flow (sum of April through
June) ################


Q_spring=features$Jan$q_maf

par(mfrow=c(1,1))
```

```
hist(Q_spring, xlab='Flow [MAF]', main='April-June')


sum_forecast=list()

# find index of features you want to keep

keep_index=which(colnames(features$Jan) %in% c('amo', 'pdo', 'enso', 'tmax_C', 'win.pcp_in'))


for (i in 1:length(features)){ # loop through features for different lead times

  df=data.frame(Q_spring, features[[i]][,keep_index])
  Sum_mod=gcmr(Q_spring~., data=df,marginal = Gamma.marg, cormat=ind.cormat, model=T)

  yhat=Sum_mod$fitted.values

  sum_forecast[[list_names[i]]]=yhat # store forecast in list for each lead time

}


######### one to one plot #############

r=3
c=1

par(mfrow=c(r,c), mar=c(5,5,2,2))

for (i in 1:length(sum_forecast)){
  yhat=sum_forecast[[i]]
  plot(x=yhat, y=Q_spring, xlab='Forecast', ylab='Observed', main='April-June')
  abline(a=0, b=1, col='red')
  rmse=sqrt(mean((yhat-Q_spring)^2))
  corr=cor(yhat, Q_spring)
  legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                                 paste('cor=',round(corr,2), sep='')), cex=1.1)

}
```

P3: non-stationary HMM

```
# CVEN 6833
# HW3
# Problem 3: Hidden Markov Model
# Nathan Bonham
# Fall 2020

rm(list=ls())

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW3")

source('HW3 Library.R')

################ Streamflow Forecasts with Hidden Markov Model
################################

# For the spring
# streamflow in problem 1 for April 1st lead time fit a HMM and make the forecast.


Apr_features=features$Apr
Q=Apr_features$q_maf
Apr_features=Apr_features[,which(colnames(Apr_features) %in% c('amo', 'pdo', 'enso',
'tmax_C', 'win.pcp_in'))]


######## Step 1: Fit a Hidden Markov Model

family <- "gamma"  # underlying distribution for hmm
discrete <- FALSE


aic1=c()


## Fit HMM models of orders 2 through 6. Obtain the AIC  for each
## They are stored in arrays aic1 and bic1
## Spit it out and the best order is the one with the least value of AIC .
## For Colorado flows it will be order 2

x=Q

for(imodel in 1:6){

  m <- imodel      #model order to fit

  stationary <- F   # use a stationary distribution of mixtures
  # different initial condition types when family == "norm"
  ic <- "same.sd"#c("same.sd","same.both","both.diff")
```

```
  fd.name <- ifelse(family == "norm", "normal", family)

  # T.P.M.
  Pi <- Pi_init(m)
  # Other ways to specify Pi
  #Pi <- diag(rep(1-.05*(m-1),m))
  #Pi[Pi == 0] <- .05
  # random guess for the transition matrix
  #Pi <- matrix(runif(m^2),m)
  #Pi <- Pi/rowSums(Pi)
  delta <- delta_init(m)

  pars <- get.named.parlist(x, m, fd.name, lower=.0, ic)#,start=list(shape1=2,shape2=2))

  # set up the model
  hmm <- dthmm(x, Pi=Pi, delta=delta, family, pars, nonstat=!stationary, discrete = discrete)

  if(imodel < 2){
    hmm <- BaumWelch(hmm, bwcontrol(maxiter = 1000,posdiff=TRUE,converge =
expression(diff > tol)))
  } else {

    hmm <- BaumWelch(hmm, bwcontrol(maxiter = 1000, tol = 1e-08))

  }


  # get the hidden states from the fitted model
  # Global decoding
  # To get the probability of being in a state: hmm$u
  decoding <- Viterbi(hmm)


  # get AIC

  aic=AIC_hmm(hmm, k=2)

  aic1=c(aic1,aic)

}

## Get the best order
bestorder = order(aic1)[1]

## Fit the model for this best order
###### For Colorado River Flows the parameters for bestorder model of 2
### obtained below will be same as what is reported in Bracken et al. (2015) paper

m <- bestorder      #model order to fit
 m<-2 # overwrite to m=2 because m =1 is boring.
stationary <- F   # use a stationary distribution of mixtures
```

```
# different initial condition types when family == "norm"
ic <- "same.sd"#c("same.sd","same.both","both.diff")

fd.name <- ifelse(family == "norm", "normal", family)

# T.P.M.
Pi <- Pi_init(m)

delta <- delta_init(m)

pars <- get.named.parlist(x, m, fd.name, lower=.0, ic)#,start=list(shape1=2,shape2=2))

# set up the model
hmm <- dthmm(x, Pi=Pi, delta=delta, family, pars, nonstat=!stationary, discrete = discrete)
#sink(tempfile())

hmm <- BaumWelch(hmm, bwcontrol(maxiter = 1000, tol = 1e-08))


# get the hidden states from the fittet model
# Global decoding
# To get the probability of being in a state: hmm$u

##the vector decoding gives the best estimate of the hidden state
## for each observation

decoding <- Viterbi(hmm)

# Gives all the model parameters
print(summary(hmm))

cat('Model order:',m,'\n')


p <- ggplot_stationary_hmm(hmm,.5)
print(p)



######## Step 2: Fit Logistic Regression to state sequence
# use previous state and features as covariates

allY=ifelse(decoding==1, 0, 1) # need 0 or 1 for GLM_fit function. 0 means state 1, 1 means
state 2
S_1=allY[-length(allY)] # previous state
Y=allY[-1]
X=data.frame(S_1, Apr_features[-1,])
glm_mod=GLM_fit(Y, X, family='binomial')
summary(glm_mod)
```

```
## alternatively, use all predictors used in problems 1 and 2

# YX=data.frame(Y, S_1, Apr_features[-1,])
#
# glm_mod=glm(Y ~., data=YX, family = binomial(link='logit'))
# summary(glm_mod)

########## Step 3: Using the glm, obtain probabilities of the states for each year

state_2prob=predict.glm(glm_mod, type='response') # recall that 0 is state 1, 1 is state 2
state_1prob=1-state_2prob
state_probs=data.frame(s1=state_1prob, s2=state_2prob)

########## Step 4: using state_probs, simulate flow from the corresponding state PDFs, to
obtain an ensemble

set.seed(5)

nsim=250
states=c(1,2)

Q_mean=rep(NA, nrow(state_probs))
Q_median=rep(NA, nrow(state_probs))
terc_mat=matrix(NA, nrow=nrow(state_probs), ncol=3)

for (i in 1: nrow(state_probs)){

  sim_vec=rep(NA, nsim)

  for (s in 1:nsim){
    state=sample(states, size=1, prob=state_probs[i,])

    if (state==1){
      sim_vec[s]= rgamma(1, shape=hmm$pm$shape[1], rate = hmm$pm$rate[1])  #simulate
from distribution for state 1 with params in hmm object
    } else { # state 2
      sim_vec[s]= rgamma(1, shape=hmm$pm$shape[2], rate = hmm$pm$rate[2])   #simulate
from distribution for state 2 with params in hmm object
    }

  }

  p1=sum(sim_vec < Qterc[1])/length(sim_vec)
  p3=sum(sim_vec > Qterc[2])/length(sim_vec)
  p2=1-(p3+p1)
  p=c(p1,p2,p3)

  terc_mat[i,]=p
  Q_mean[i]=mean(sim_vec)
  Q_median[i]=median(sim_vec)
```

```
}


################## Step 5: plot historic vs ensemble mean forecast along with 1:1 line.
###################
###### calculate metrics RPSS and correlation


par(mfrow=c(1,2))

plot(x=Q_mean, y=Q[-1], xlab='forecast [MAF]', ylab='Observed [MAF]', main='Mean')
abline(a=0, b=1, col='red')
rmse=sqrt(mean((Q_mean-Q[-1])^2))
rpss=rps(obs=historical_climate[-1], pred=terc_mat, baseline=climatology)$rpss
corr=cor(Q_mean, Q[-1])
legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                               paste('cor=',round(corr,2), sep=''),
                               paste('rpss=', round(rpss,2), sep='')), cex=1.1)

plot(x=Q_median, y=Q[-1], xlab='forecast [MAF]', ylab='Observed [MAF]', main='Median')
abline(a=0, b=1, col='red')
rmse=sqrt(mean((Q_median-Q[-1])^2))
rpss=rps(obs=historical_climate[-1], pred=terc_mat, baseline=climatology)$rpss
corr=cor(Q_median, Q[-1])
legend('bottomright', legend=c(paste('rmse=',round(rmse, 2), sep=''),
                               paste('cor=',round(corr,2), sep=''),
                               paste('rpss=', round(rpss,2), sep='')), cex=1.1)
```

P4: non-stationary EVA

```
# CVEN 6833
# HW3
# P4: Nonstationary Extreme Value Analysis
# Nathan Bonham



############################################### Nonstationary EVA
####################

# 4. Perform a space-time nonstationary EVA on the winter 3-day maximum precipitation.
# Below are the steps:
#   . At each location fit a nonstationary GEV model to the 3-day winter maximum precipitation
as a
# function of the 3 covariates - the leading ~3 winter SST PCs. Make only the location
parameter
# of the GEV non-stationary. This will result in 4 coefficients (intercept plus the three covariates)
# . Fit a spatial model to each of the coefficients and to the scale and shape parameter
# . For couple of wet and dry years (you can select the years based on the average spatial 3-
day
#                                 precipitation) - obtain the 2-year, 50-year and 100-year return levels on
the spatial grid
# o Using the spatial models obtain the GEV parameters at each grid point
# o Estimate the 2-year, 50-year and 100-year return levels at each grid point and map
# them. Compare with 2-year return levels with the observed values in the selected years
# . For couple of representative locations plot the time series of 3-day precipitation maximum
# along with the time varying return levels. Compare them with the stationary return levels


rm(list=ls())

time_elapsed=0


setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced Stats/HW/HW3")#load
the libary file
suppressPackageStartupMessages(source("Alvaro Lib.R"))
source("Alvaro Lib.R")

source('HW3 Library.R')

################################### Read data ###################################

setwd("G:/My Drive/CU Boulder/Classes/20-21/CVEN 6833 Advanced
Stats/HW/HW3/data")#load the libary file

#Max winter precipitation
data=read.delim("Max_Winter_Seas_Prec.txt", header = TRUE, sep = "\t", dec = ".")#in mmm
#Total summer precipitation
```

```
data1=read.delim("Total_Prec.txt", header = TRUE, sep = "\t", dec = ".")#in mmm
#Stations information (lat,long, elev, etc...)
info=read.delim("station_location_swus.txt", header = TRUE, sep = "\t", dec = ".")



#load covariates

sst_data=readSST()
sst=sst_data$sst
years=1901:2018 # years of SST data

sst1964=sst[which(years==data$YEAR[1]):nrow(sst),] # truncate SST data to match precip data

sst_pca=pca(sst1964, scale = T)
z_pca=sst_pca$pcs[,1:3]

elev_grid=read.delim("Elevation_grid_1deg.txt", header = TRUE, sep = "\t", dec = ".")

datafit=data.frame(t=data$YEAR , Z1=z_pca[,1], Z2= z_pca[,2], Z3=z_pca[,3])

############################## Fit a non stationary GEV for each location considering only
the location parameter as nonstationary ###############

ptm = proc.time()
name_par=c("Station","mu_0","mu_1","mu_2","mu_3","sigma","xi")
params=data.frame(matrix(data=-NA,nrow = dim(data)[2]-1,ncol=length(name_par)))#to save
the GEV parameter of each station
params[,1]=info$Station
colnames(params)=name_par
gev_ob=list(desc="")

for (i in 2:dim(data)[2]) { # looping through locations
  # print(i-1)

  fit.gev= fevd(data[,i],datafit,location.fun=~Z1+Z2+Z3,use.phi = TRUE)
  gev_ob[[paste("Est_",i-1,sep = "")]]=fit.gev
  params[i-1,2:dim(params)[2]]=fit.gev$results$par
}
time_elapsed=time_elapsed+(proc.time() - ptm)[3]



ptm = proc.time()
X=info[,c(4,3,5)]# long, lat, and elev
X1=elev_grid #long, lat, and elev of 1deg grid
n.samples=1000#the number of MCMC iterations
burn=0.5#numner of samples discarded, warmup
thin1=1#sample thinning factor, if thin = 10 then 1 in 10 samples are considered between start
and end
displ=TRUE
```

```
thin1=1
plots=list(desc="plots")#to save posterior distribution plots
par=list(desc="parameter values")#to save the postrior vaues of the GEV coefficients over the
grid
acep_rate=c()#to save the aceptance_rate for each coefficient simulated

# markov chain monte carlo

for (i in 2:dim(params)[2]) {
  Y=params[,i]
  result=get.samples_1(Y,X,X1,colnames(params)[i],n.samples,burn,thin1,displ=FALSE)
  plots[[paste("Dist",(i-1),sep = "")]]=result$p
  par[[paste("Median_par",(i-1),sep = "")]]=result$Pred
  par[[paste("par",(i-1),sep = "")]]=result$y
  acep_rate=c(acep_rate,result$acep_r[2])
}
time_elapsed=time_elapsed+(proc.time() - ptm)[3]
sprintf("Aceptance rate for each coefficients are (percent): %s ",acep_rate)


sprintf("elapsed time: %s minutes",round((proc.time() - ptm)[3]/60,2))


########################################## Histograms of GEV coefficients
#######################################

name=colnames(params)[2:dim(params)[2]]
p1=ggplot() +
  geom_histogram(aes(as.vector(params[,2]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[1],'$',sep = "")),
       x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

p2=ggplot() +
  geom_histogram(aes(as.vector(params[,3]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[2],'$',sep = "")),
       x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

p3=ggplot() +
  geom_histogram(aes(as.vector(params[,4]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[3],'$',sep = "")),
       x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

p4=ggplot() +
  geom_histogram(aes(as.vector(params[,5]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[4],'$',sep = "")),
```

```
      x = "Predictions")+
  #theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

p5=ggplot() +
  geom_histogram(aes(as.vector(params[,6]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[5],'$',sep = "")),
      x = "Predictions")+
  # theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))

p6=ggplot() +
  geom_histogram(aes(as.vector(params[,7]),..density..),fill="gray50",col='black') +
  labs(title = TeX(paste('Histogram of $\\',name[6],'$',sep = "")),
      x = "Predictions")+
  # theme_light()+
  theme(plot.title = element_text(size = 13, face = "bold", hjust = 0.5))


multiplot(p1,p2,p3,p4,p5,p6, cols = 3)

multiplot(plots$Dist1,plots$Dist2,plots$Dist3,plots$Dist4,plots$Dist5,plots$Dist6, cols = 3)

################### Spatial maps of posterior params, GEV fitted params, and standard error
#############

ygrid=seq(31.5,42,by=1)
# ygrid_1=seq(range(ygrid)[1],range(ygrid)[2],by=res_grid)
ny=length(ygrid) # number of latitudinal locations

xgrid=seq(-114.5,-102,by=1)
# xgrid_1=seq(range(xgrid)[1],range(xgrid)[2],by=res_grid)
nx=length(xgrid) # number of longitudinal locations

nglobe = nx*ny
# creation of spatial grid
xygrid=matrix(0,nrow=nglobe,ncol=2)
i=0
for(iy in 1:ny){
  for(ix in 1:nx){
    i=i+1
    xygrid[i,2]=ygrid[iy]
    xygrid[i,1]=xgrid[ix]
  }
}

#find the locations in the rectangular grid
index1=c()
for (i in 1:nrow(X1)) {
  tmp=which(X1[i,1]==xygrid[,1] & X1[i,2]==xygrid[,2])
  index1=c(index1,tmp)
```

```
}


################ Spatial Bayesian model ############################

r_pal=rev(brewer.pal(10,"RdBu"))
MainStates <- map_data("state")
xlim1=c(-115,-101.3)
ylim1=c(29.5,42.5)

namex=seq(from=-114,to=-102,by=2)
labx=paste(abs(namex[1]),"\u00B0W",sep = "")
for (i in 2:length(namex)) {
  labx=c(labx,paste(abs(namex[i]),"\u00B0W",sep = ""))
}
namey=seq(from=30,to=42,by=2)
laby=paste(abs(namey[1]),"\u00B0N",sep = "")
for (i in 2:length(namey)) {
  laby=c(laby,paste(abs(namey[i]),"\u00B0N",sep = ""))
}

for (i in 2:dim(params)[2]) {
  Y=params[,i]
  plot_sp_grid(par[[paste("Median_par",(i-1),sep =
""")]],Y,X[,1:2],xgrid,ygrid,nx*ny,index1,colnames(params)[i],xlim1,ylim1,namex,namey,labx,laby,
r_pal)
}



############ Estimate 2-year , 50-year, and 100-year return level of max seasonal precipitation
for a wet year #############

year_avg=apply(data[,-1], 1, mean)

ind_pc_max=which.max(year_avg)#pick a wet year
ind_pc_min=which.min(year_avg)#pick a dry year

ind_pc=ind_pc_min # choose max or min to plot

sprintf("The wet year is: %s",data$YEAR[ind_pc_max])
sprintf("The dry year is: %s",data$YEAR[ind_pc_min])

ptm = proc.time()
period=c(2,50,100)# return periods considered
set.seed(3000)
# xygrid=elev_grid$elev_data[,1:2]
nsim=ncol(par$par1)#number of samples
estimate=list(T2=matrix(NA,nrow(par$par1),ncol(par$par1)),T50=matrix(NA,nrow(par$par1),nco
l(par$par1)),T100=matrix(NA,nrow(par$par1),ncol(par$par1)))
```

```
for (sim in 1:nsim) {
  par1=data.frame(matrix(data=NA,nrow = dim(X1)[1],ncol=dim(params)[2]-1))#to save the GEV
coefficients of the grid
  for (i in 1:(dim(params)[2]-1)) {
    par1[,i]=par[[paste("par",(i),sep = "")]][,sim]
  }

  for (i in 1:nrow(X1)) {
    loc=as.matrix(datafit[ind_pc,2:4]) %*% as.numeric(par1[i,2:4])+as.numeric(par1[i,1])
    loc=as.numeric(loc)
    scal=exp(as.numeric(par1[i,5]))
    shap=as.numeric(par1[i,6])
    tmp1=as.numeric(qgev(1-1/period, xi = shap, mu = loc, beta = scal, lower.tail =
TRUE))#column 1: 2yr, column 2:50yr, column 3: 100yr
    estimate$T2[i,sim]=tmp1[1]
    estimate$T50[i,sim]=tmp1[2]
    estimate$T100[i,sim]=tmp1[3]

  }
}
time_elapsed=time_elapsed+(proc.time() - ptm)[3]
sprintf("elapsed time: %s minutes",round((proc.time() - ptm)[3]/60,2))


####################plot of the median for T=2yr and T=100yr


med_predT2=apply(estimate$T2, 1, median)
med_predT50=apply(estimate$T50, 1, median)
med_predT100=apply(estimate$T100, 1, median)

Y=as.numeric(data[ind_pc,2:ncol(data)])
par( oma=c(0,0,3,1)) # save some room for the legend
par(mfrow = c(2, 2))
par(mar = c(4.5, 4.7, 2, 1.5))

zlim=quantile(Y, c(0, .90))

# plot of the observed precip for the wet year
quilt.plot(X[,1],X[,2],Y,axes=F,xlab="Longitude",ylab="Latitude",main="Observed",
zlim=zlim,ylim=ylim1,xlim=xlim1,col=r_pal,cex.main=1.4,cex.lab=1.3, nx=32, ny=32)
US(add=TRUE, col="gray30", lwd=2)
box()
axis(1, at=namex, labels=labx)
axis(2, at=namey, labels=laby)

zlim=range(med_predT2)

zfull = rep(NaN,nglobe)
zfull[index1]=med_predT2
```

```
zmat = matrix(zfull,nrow=length(xgrid),ncol=length(ygrid))
image.plot(xgrid,ygrid,zmat,ylim=ylim1,xlim=xlim1,main ="Median T=2 yrs",
zlim=zlim,axes=F,xlab="Longitude",ylab="Latitude",cex.main=1.4,cex.lab=1.3,col=r_pal)
US(add=TRUE, col="gray30", lwd=2)
box()
axis(1, at=namex, labels=labx,cex.axis=1)
axis(2, at=namey, labels=laby,cex.axis=1)

zlim=range(med_predT50)
zfull = rep(NaN,nglobe)
zfull[index1]=med_predT50
zmat = matrix(zfull,nrow=length(xgrid),ncol=length(ygrid))
image.plot(xgrid,ygrid,zmat,ylim=ylim1,xlim=xlim1, zlim=zlim,main ="Median T=50
yrs",axes=F,xlab="Longitude",ylab="Latitude",cex.main=1.4,cex.lab=1.3,col=r_pal)
# contour(xgrid,ygrid,zmat,ylim=ylim1,xlim=xlim1,add=TRUE,nlev=4,lwd=1)
US(add=TRUE, col="gray30", lwd=2)
# grid(col="gray30",lty=2)
box()
axis(1, at=namex, labels=labx,cex.axis=1)
axis(2, at=namey, labels=laby,cex.axis=1)

zlim=range(med_predT100)
zfull = rep(NaN,nglobe)
zfull[index1]=med_predT100
zmat = matrix(zfull,nrow=length(xgrid),ncol=length(ygrid))
image.plot(xgrid,ygrid,zmat,ylim=ylim1,xlim=xlim1, zlim=zlim,main ="Median T=100
yrs",axes=F,xlab="Longitude",ylab="Latitude",cex.main=1.4,cex.lab=1.3,col=r_pal)
# contour(xgrid,ygrid,zmat,ylim=ylim1,xlim=xlim1,add=TRUE,nlev=4,lwd=1)
US(add=TRUE, col="gray30", lwd=2)
# grid(col="gray30",lty=2)
box()
axis(1, at=namex, labels=labx,cex.axis=1)
axis(2, at=namey, labels=laby,cex.axis=1)


mtext(paste("Winter Season 3-day Max. Prec. for year ",data$YEAR[ind_pc]," (mm)",sep = ""),
outer = TRUE, side = 3, cex = 1.3, line = 1)


############ For couple of representative locations plot the time series of 3-day precipitation
maximum
# along with the time varying return levels. Compare them with the stationary return levels

# hand select 3 locations with interesting return periods
locs=c(47, 3, 70) # these are the indexes in the lat/lon grid in X


rp=c(2,50,100) # return periods

T.df=matrix(NA, nrow=nrow(datafit), ncol=(length(rp)+1)) # preallocate a dataframe for precip
values
```

```
T.df[,1]=datafit$t
names=1:(length(rp)+1)
names[1]='year'

for (i in 1:(length(rp)+1)){ # create column names of form T2, T100, etc

  if (i ==1){
  # nothing, skip year column
  } else {
    names[i]=paste('T',rp[i-1], sep='')
  }
}

T.df=data.frame(T.df)
colnames(T.df)=names

locs_T=list() # preallocate a list to store data frame of return levels

c=1
for (i in locs){ # loop through locations
  work.df=T.df
  p=params[i,-1]

  for(t in 1:nrow(datafit)){ # loop through years

    Xloc=datafit[t,-1]
    loc=p[1]+sum(Xloc*p[2:4])
    scale=exp(p$sigma)
    shape=p$xi

    for(z in 1:length(rp)){ # loop through return levels
      prob=1/rp[z]
      work.df[t,(z+1)]=qevd(prob, loc=loc, scale=scale, shape=shape, type='GEV', lower.tail = F) #
place values in dataframe
    }
  }
  locs_T[[c]]=work.df # store data frame in a list for each location
  c=c+1
}



site=1:length(locs)

c=1
for (i in locs){
  site[c]=paste('lon ', X$Long[i],',', 'lat ', X$Lat[i], ',' ,'elev ', round(X$Elev[i]), 'm', sep='')
  c=c+1
}

par(mfrow=c(1,3))
```

```
col_vec=c('blue', 'green', 'red')


for(i in 1:length(locs)){

  col=locs[i]+1

  y=data[,col]

  plot(x=data$YEAR, y=y, type='l', main=site[i], xlab='Year', ylab='Precip [mm]', ylim=c(0,
2.2*max(data[,col])))

  for (c in 2:ncol(work.df)){
    perc=1-1/rp[c-1]
    q=quantile(y, perc)
    lines(x=data$YEAR, y=rep(q, length(data$YEAR)), col='lightgrey')
    lines(x=data$YEAR, y=locs_T[[i]][,c], col=col_vec[c-1], lty=2)
  }

  legend('topright', legend=c('T2', 'T50', 'T100', 'Observed', 'Constant Levels'),
         col=c(col_vec, 'black', 'lightgrey'), lty=c(2,2,2,1,1), cex=.85)

}
```