Neural network examples in R

1 Ozone data

We apply the neural networks to the `ozone` data which was analyzed before using the `nnet` package, due to Venables and Ripley (2002). We start with just three variables for simplicity and fit a feed-forward neural network with one hidden layer containing two units and a linear output unit.

```
> library(faraway)
> library(nnet)
> attach(ozone)
> nnmdl <- nnet(O3 ~ temp + ibh + ibt, ozone, size=2, linout=T)

# weights:  11
initial  value 66652.717816
final  value 21115.406061
converged

> sum((O3-mean(O3))^2)

[1] 21115.41
```

The RSS of 21,115 is equal to $\sum_i (y_i - \bar{y})^2$, so the fit is not any better than the null model (fit with a constant). If you repeat this, your result may differ slightly because of the random starting point. The problem comes from the initial selection of the weights. It is hard to select the initial weights when the variables have very different scales. The solution is to rescale the data to mean zero and unit variance.

```
> apply(ozone,2,sd)

        O3          vh        wind    humidity        temp         ibh
  8.011277  105.708241    2.116963   19.865000   14.458737 1803.885870
       dpg         ibt         vis         doy
 35.717181   76.679424   79.362393  104.376374

> ozone2 <- scale(ozone)
```

Since neural network uses random initial weights, the algorithm may not give the same results for each replication. So we try to refitting the model 100 times using different initial weights and find the best fit of these 100 attempts.

```
> bestrss <- 10000
> for (i in 1:100){
   set.seed(i)
   nnmdl <- nnet(O3 ~ temp + ibh + ibt, ozone2, size=2, linout=T, trace=F)
   if (nnmdl$value < bestrss){
```

```
    bestnn <- nnmdl
    bestrss <- nnmdl$value
  }
 }
> bestnn$value

[1] 88.53719

> summary(bestnn)

a 3-2-1 network with 11 weights
options were - linear output units
  b->h1  i1->h1  i2->h1  i3->h1
 -56.94   29.52 -100.00  -55.71
  b->h2  i1->h2  i2->h2  i3->h2
   1.14   -0.95    0.83    0.28
   b->o   h1->o   h2->o
   3.37   -0.69   -4.52
```

The criterion function has 11 parameters or weights. The notation `i2->h1`, for example, refers to the link between the second input variable and the first hidden neuron. `b` refers to the bias, which takes a constant value of one. We see that there is one skip-layer connection, `b->o`, from the bias to the output.

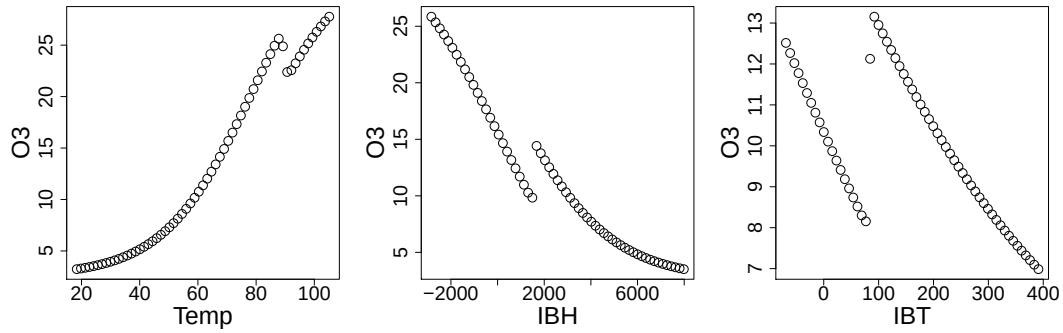
Comparing to statistical models, the parameters of an neural network are not interpretable. In addition, NNs are not based on a probability model that expresses the structure and variation. As a result, they have no standard errors. The R^2 for the fit is:

```
> 1-bestnn$value/329 #ozone2 is standardized

[1] 0.73089
```

which is slightly higher than the GAM, whose R^2 is around 71%.

Although the NN weights may be difficult to interpret, we can get some sense of the effect of the predictors by observing the marginal effect of changes in one or more predictor as other predictors are held fixed at their mean values. Since the data has been standardized for NN fitting, we need to restore the original scales.

```
> ozmeans <- attributes(ozone2)$"scaled:center"
> ozscales <- attributes(ozone2)$"scaled:scale"
> par(mfrow=c(1,3),mar=c(5,5,2,2),cex.lab=2.5,cex.axis=2)
> xx <- expand.grid(temp=seq(-3,3,0.1),ibh=0,ibt=0)
> plot(xx$temp*ozscales['temp']+ozmeans['temp'],
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="Temp",ylab="O3")
> xx <- expand.grid(temp=0,ibh=seq(-3,3,0.1),ibt=0)
> plot(xx$ibh*ozscales['ibh']+ozmeans['ibh'],
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="IBH",ylab="O3")
> xx <- expand.grid(temp=0,ibh=0,ibt=seq(-3,3,0.1))
> plot(xx$ibt*ozscales['ibt']+ozmeans['ibt'],
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="IBT",ylab="O3")
```



We see some surprising discontinuities in the plots which do not seem consistent with what we might expect for the effect of these predictors. This situation is analogous to the multicollinearity problem in linear regression where unreasonably large regression coefficients are often seen. The NN tends to choose large weights in order to optimize the fit, but the predictions will be unstable, especially for extrapolation.

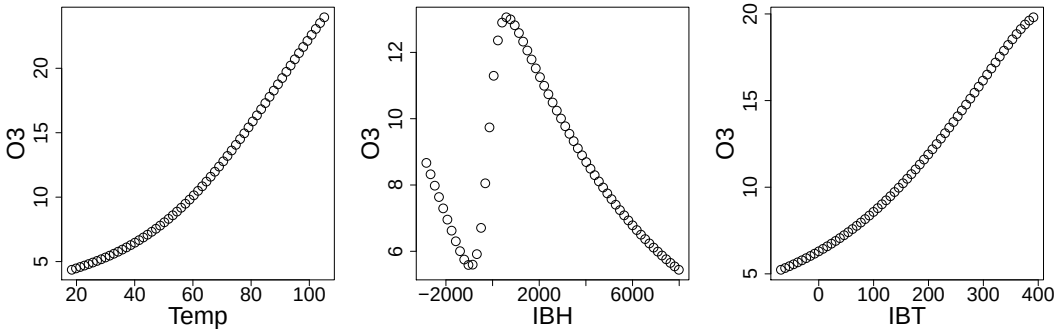
We can put a penalty such as weight decay to obtain a more stable fit. Let's try $\lambda = 0.001$:

```
> bestrss <- 10000
> for (i in 1:100){
   set.seed(i)
   nnmdl <- nnet(O3 ~ temp + ibh + ibt, ozone2, size=2, linout=T,
   decay=0.001, trace=F)
   if (nnmdl$value < bestrss){
     bestnn <- nnmdl
     bestrss <- nnmdl$value
   }
 }
> bestnn$value

[1] 92.05528

> summary(bestnn)

a 3-2-1 network with 11 weights
options were - linear output units  decay=0.001
 b->h1 i1->h1 i2->h1 i3->h1
-11.18  -2.07  -7.32   2.85
 b->h2 i1->h2 i2->h2 i3->h2
 -1.18   0.60  -0.42   0.45
  b->o  h1->o  h2->o
 -1.11  -1.39   4.05
```

We can see the value of the best RSS is somewhat larger than before. This is expected because weight decay sacrifices some fit to the current data to obtain a more stable result. We repeat the assessment of the marginal effects as above.

```
> par(mfrow=c(1,3),mar=c(5,5,2,2),cex.lab=2.5,cex.axis=2)
> xx <- expand.grid(temp=seq(-3,3,0.1),ibh=0,ibt=0)
> plot(xx$temp*ozscales['temp']+ozmeans['temp'],
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="Temp",ylab="O3")
> xx <- expand.grid(temp=0,ibh=seq(-3,3,0.1),ibt=0)
> plot(xx$ibh*ozscales['ibh']+ozmeans['ibh'],
```

```
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="IBH",ylab="O3")
> xx <- expand.grid(temp=0,ibh=0,ibt=seq(-3,3,0.1))
> plot(xx$ibt*ozscales['ibt']+ozmeans['ibt'],
 predict(bestnn,new=xx)*ozscales['O3']+ozmeans['O3'],cex=2,xlab="IBT",ylab="O3")
```



We see the fits are now smoother. Note that `ibh` is strictly positive in practice so the strange behavior on the negative part is irrelevant. Comparing to the GAM, the shapes are similar for `temp` and `ibh`. But the `ibt` plots looks quite different.

Now let's look at the full dataset. We use four hidden units since there are now more inputs.

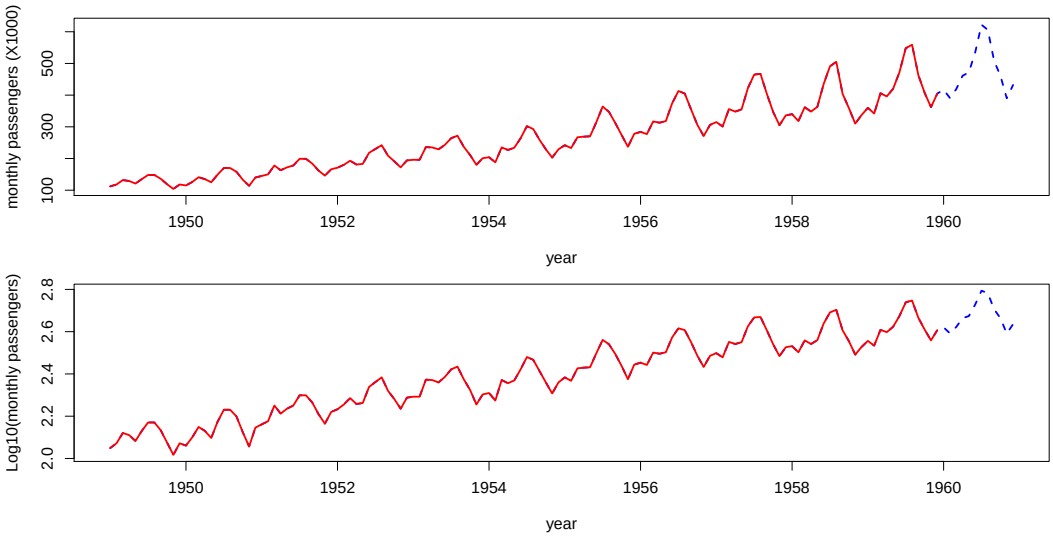
```
> bestrss <- 10000
> for (i in 1:100){
   set.seed(i)
   nnmdl <- nnet(O3 ~ ., ozone2, size=4, linout=T, trace=F)
   if (nnmdl$value < bestrss){
     bestnn <- nnmdl
     bestrss <- nnmdl$value
   }
 }
> bestnn$value

[1] 49.57413

> 1 - bestnn$value/329

[1] 0.8493188

> detach(ozone)
```

The fit is good and the R^2 is substantially higher than the linear and single tree model fits. It is slightly higher than the GAM fit using the full dataset. But the additive model has the interpretability that the NN lacks. Finally the MARS model (taught in EXST7152) fits better and was also interpretable.

2 Airline passenger example

This is a time series data, listed by Box *et al.* (1994) and earlier by Brown (1962). The following figure shows that the data have an upward trend together with seasonal variation whose size is roughly proportional to the local mean level (called *multiplicative* seasonality).

```
> loc<-"C:/Users/bli/Desktop/data-mining/dataset/airpass.dat"
> airpass <- scan(loc)
> years   <- seq(1949,1960+11/12,by=1/12)
> log.airpass <- log10(airpass)
> par(mfrow=c(2,1),mar=c(4,4,1,1),cex.lab=1.1, cex.axis=1.1,cex.main=1.5)
> plot(years, airpass, type="l", col="blue",lty=2,lwd=2,
      xlab="year", ylab="monthly passengers (X1000)")
> lines(years[1:132], airpass[1:132],col="red",lwd=2)
> plot(years, log.airpass, type="l", col="blue",lty=2,lwd=2,
      xlab="year", ylab="Log10(monthly passengers)")
> lines(years[1:132], log.airpass[1:132],col="red",lwd=2)
```



A common approach to dealing with this type of seasonality is to choose an appropriate transformation, usually logarithms, to make the seasonality *additive*. The standard Box-Jenkins analysis involves taking logarithms of the data followed by seasonal and non-seasonal differencing to make the series stationary. A special type of seasonal autoregressive integrated moving average (SARIMA) model is then fitted here. We use the first 11 years data to fit the model and then making forecasts of the last 12 monthly observations.

```
> sarima.model <- arima(log.airpass[1:132], order=c(1,1,0),
                        seasonal=list(order=c(1,0,0), period=12))
> sarima.model

Call:
arima(x = log.airpass[1:132], order = c(1, 1, 0), seasonal = list(order = c(1,
    0, 0), period = 12))

Coefficients:
          ar1    sar1
      -0.2665  0.9298
s.e.   0.0866  0.0233
```

```
sigma^2 estimated as 0.0003299:  log likelihood = 327.19,  aic = -648.38

> pred1 <- predict(sarima.model,12)$pred #prediction on next 12 pts
> mean((10^pred1-airpass[133:144])^2)

[1] 921.9631

> 1-sum((10^pred1-airpass[133:144])^2)/sum((airpass[133:144]-mean(airpass[133:144]))^2)

[1] 0.8335648
```

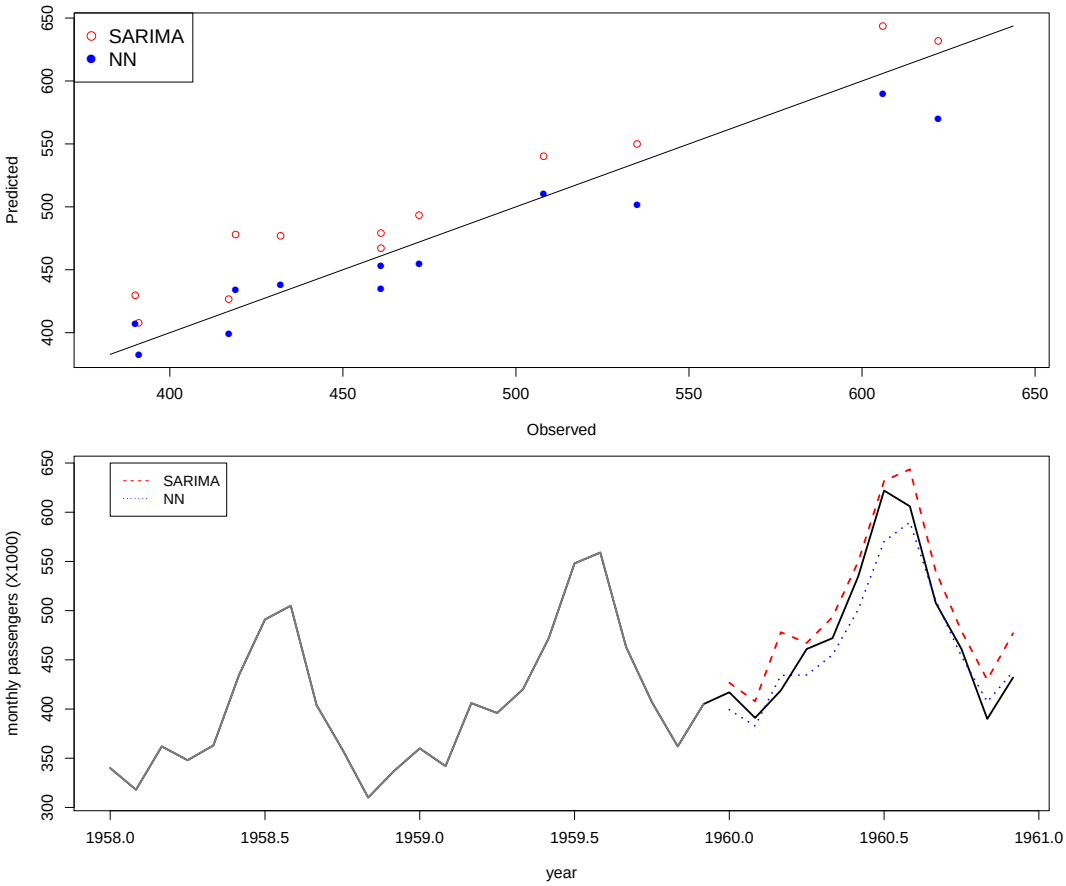
We can also apply the neural network on this time series data. Due to its seasonality nature, the value at time t is to be forecasted using only the values at lags 1 (i.e. last month's value) and 12 (i.e. the value in the same month from last year). For single-layer neural networks, there are two tuning parameters: number of hidden units and weight decay. To get an optimal network, we need to search the optimal values for these two parameters with a grid search. In `R`, there is an excellent package `caret` (classification and regression training) which contains functions to streamline the model training process for many complex regression and classification problems. The `train` function set up a grid search on tuning parameters for various of data mining methods. Here we borrow the strength of `train` function in `caret` to run the `nnet` function. For details, see `http://topepo.github.io/caret/bytag.html` *Shall we scale the airline data here before we apply the nn?*

```
> library(caret)
> dat1 <- data.frame(y=log.airpass[1:132],x1=c(NA,log.airpass[1:131]),
                     x2=c(rep(NA,12),log.airpass[1:120]))
> dat2 <- data.frame(y=log.airpass[133:144],x1=log.airpass[132:143],
                     x2=log.airpass[121:132])
> set.seed(2)
> nn.model <- train (y ~ x1 + x2,dat1,method = "nnet",linout = TRUE,trace = FALSE)
```

We can predict in two ways. *The first one is kind of unfair. Why?* We see that neural network performs better than the traditional SARIMA model.

```
> pred2 <- predict (nn.model, dat2);
> mean( (10^pred2 - airpass[133:144])^2 )

[1] 433.8942

> pred3 <- rep(0,12)
> pred3[1] <- predict(nn.model,dat2[1,])
> for (i in 2:12){
   pred3[i] <- predict(nn.model,data.frame(x1=pred3[(i-1)],x2=dat2$x2[i]))
 }
> mean((10^pred3 - airpass[133:144])^2)

[1] 504.8303

> 1-sum((10^pred3-airpass[133:144])^2)/sum((airpass[133:144]-mean(airpass[133:144]))^2)

[1] 0.9088667
```

```
> lim <- range(c(airpass[133:144],10^pred1,10^pred3))
> par(mfrow=c(2,1),mar=c(4,4,1,1),cex.lab=1.1, cex.axis=1.1,cex.main=1.5)
> plot(airpass[133:144],10^pred1,xlab="Observed",ylab="Predicted",col="red",pch=1,
      xlim=lim,ylim=lim)
> points(airpass[133:144],10^pred3,col="blue",pch=16)
> lines(lim,lim)
> legend("topleft",legend=c("SARIMA","NN"),pch=c(1,16),col=c("red","blue"),cex=1.3)
> plot(years[109:144], airpass[109:144], type="l",
      ylim=range(c(airpass[109:132],10^pred1,10^pred2)),
      xlab="year", ylab="monthly passengers (X1000)",lwd=2)
> lines(years[109:132], airpass[109:132],col=grey(0.5),lwd=2)
> lines(years[133:144],10^pred1,col="red",lwd=2,lty=2)
> lines(years[133:144],10^pred3,col="blue",lwd=2,lty=3)
> legend(1958,650,legend=c("SARIMA","NN"),col=c("red","blue"),lty=c(2,3))
```

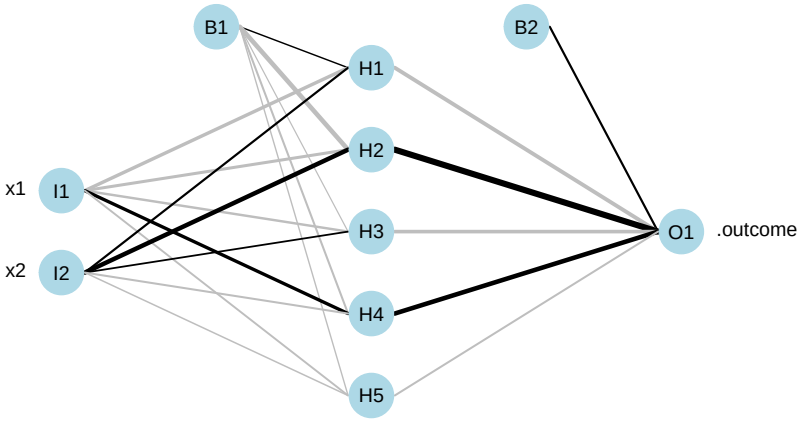


Although **nnet** package does not provide any tool to visualize the network, someone contributes the source code to do that. The final **nnet** model is saved in the **nn.model$finalModel**. The darkness and width of the lines are proportional to the absolute values of the weights.

```
> summary(nn.model$final)

a 2-5-1 network with 21 weights
options were - linear output units  decay=1e-04
 b->h1 i1->h1 i2->h1
  0.15  -1.17   0.54
 b->h2 i1->h2 i2->h2
```

```
 -1.77  -0.95   1.71
 b->h3 i1->h3 i2->h3
 -0.03  -0.66   0.32
 b->h4 i1->h4 i2->h4
 -0.46   1.13  -0.45
 b->h5 i1->h5 i2->h5
 -0.12  -0.38  -0.19
 b->o h1->o h2->o h3->o h4->o h5->o
 0.53 -1.41  2.56 -1.24  1.75 -0.44
```

```
> loc <- "C:/Users/bli/Desktop/data-mining/neural_network/code/nnet_plot_update.R"
> source(loc)
> library(reshape)
> plot.nnet(nn.model$final)
```



Remarks:

- The neural networks are commonly considered to be able to automatically fit the data well with no need to take care of the data such as transformation of input variables. In this study, we tried the NN on the raw data and see the results is worse than on the log-transformed data.

```
> #Training set: dat1; test set: dat2
> dat.1 <- data.frame(y=airpass[1:132],x1=c(NA,airpass[1:131]),
                      x2=c(rep(NA,12),airpass[1:120]))
> dat.2 <- data.frame(y=airpass[133:144],x1=airpass[132:143],
                      x2=airpass[121:132])
> set.seed(2)
> nn.model2 <- train (y ~ x1 + x2, dat.1, method = "nnet", linout = TRUE, trace = FALSE)
> pred4 <- rep(0,12)
> pred4[1] <- predict(nn.model2,dat.2[1,])
> for (i in 2:12){
```

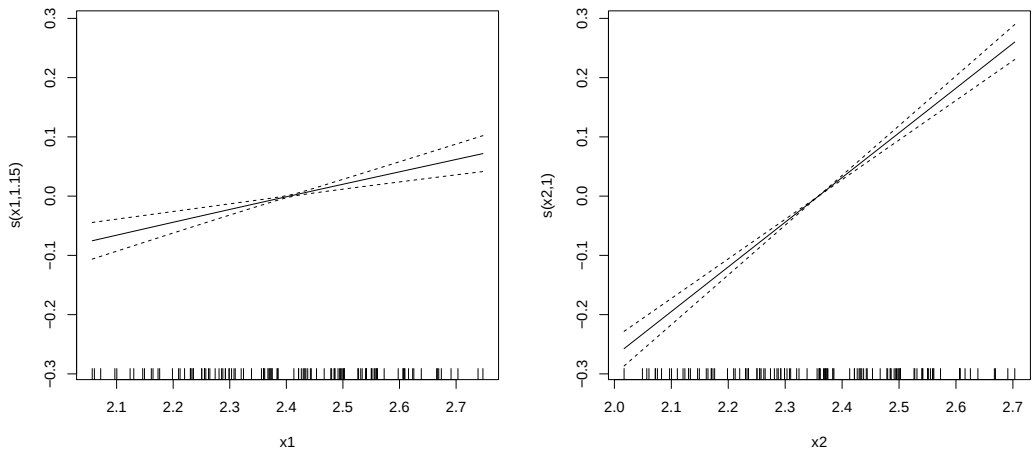
```
   pred4[i] <- predict(nn.model2,data.frame(x1=pred4[(i-1)],x2=dat.2$x2[i]))
 }
> mean((pred4 - airpass[133:144])^2)

[1] 1150.249
```

- For NN, we still need to carefully choose the input variables. Since we believe the data has seasonality with 12 months as a period, we use lag-12 value as one of the input. If we ignore the seasonality and only use lag-1 and lag-2 values as the input, then the NN results will be much worse.

```
> dat.1 <- data.frame(y=log.airpass[1:132],x1=c(NA,log.airpass[1:131]),
                      x2=c(rep(NA,2),log.airpass[1:130]))
> dat.2 <- data.frame(y=log.airpass[133:144],x1=log.airpass[132:143],
                      x2=log.airpass[131:142])
> set.seed(2)
> nn.model3 <- train (y ~ x1 + x2,dat.1,method = "nnet",linout = TRUE,trace = FALSE)
> pred5 <- rep(0,12)
> pred5[1] <- predict(nn.model3,dat.2[1,])
> pred5[2] <- predict(nn.model3,data.frame(x1=pred5[1],x2=dat.2$x2[2]))
> for (i in 3:12){
   pred5[i] <- predict(nn.model3,data.frame(x1=pred5[(i-1)],x2=pred5[(i-2)]))
 }
> mean((10^pred5 - airpass[133:144])^2)

[1] 21742.83
```

- NN models are hard to interpret. GAMs provides an alternative nonparametric approach to the identification of non-linear time series which requires less structure to be imposed on the data, and more familiar to statisticians and are arguably more helpful in understanding the nature of the relationship between the response and input variables.

```
> library(mgcv)

> gam.model <- gam(y ~ s(x1) + s(x2), data=dat1)
> pred6 <- rep(0,12)
> pred6[1] <- predict(gam.model,dat2[1,])
> for (i in 2:12){
   pred6[i] <- predict(gam.model,data.frame(x1=pred6[(i-1)],x2=dat2$x2[i]))
 }
> mean((10^pred6 - airpass[133:144])^2)

[1] 450.9895

> par(mfrow=c(1,2))
> plot(gam.model)
```



3 Ensemble of neural networks

This is a simulation study with ten input features $X_1, \ldots, X_10$ generated from independent standard normal distribution. The deterministic binary target Y is defined by

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) = 9.34 \\ 0 & \text{otherwise} \end{cases}$$

Why the cutoff is set at $\chi^2_{10}(0.5)$, the median of chi-squared distribution with 10 df? The training set has 500 observations and test set has 5,000 independent observations.

```
> N <- 5500
> set.seed(1)
> x <- matrix(rnorm(N*10),N,10); colnames(x) <- paste("X",1:10,sep="")
> z <- apply(x,1,function(x) sum(x^2))
> y <- rep(0,N); y[z>=qchisq(0.5,df=10)] <- 1
> table(y)

y
   0    1
2711 2789

> data.train <- data.frame(x[1:500,]); data.train$Y <- y[1:500]
> data.test <- data.frame(x[501:N,]); data.test$Y <- y[501:N]
```

First let's fit a neural network with optimal number of hidden units and weight decay value through grid search in `train function` in `caret` package.

```
> set.seed(1)
> nn.model <- train (Y ~ ., data.train, method = "nnet", trace = FALSE)
> pred1 <- predict (nn.model, data.test)
> tab1 <- table(pred1>=.5,data.test$Y)
> 1-sum(diag(tab1))/sum(tab1) #misclassification rate

[1] 0.3366
```

```
> summary(nn.model)

a 10-5-1 network with 61 weights
options were - decay=0.1
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1 i10->h1
   2.45    0.07    0.55   -0.18   -0.56    0.33   -1.59    0.25   -0.64   -1.85    1.22
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2 i10->h2
  -2.58   -0.03   -1.09    1.10   -0.81    0.76   -0.11   -0.12   -0.85    0.51    1.34
  b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3 i10->h3
  -2.38   -1.19    1.41   -0.65    0.83    0.14   -1.55   -1.06   -0.20   -0.01    0.73
  b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4 i10->h4
   2.11    0.38   -0.09   -0.76    0.35   -0.89   -0.23   -0.58   -0.23    1.47    1.72
  b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5 i10->h5
  -1.27    2.12    0.43   -1.31   -0.29   -1.67    0.39    0.72    0.23   -0.34    0.55
 b->o h1->o h2->o h3->o h4->o h5->o
 2.36 -3.72  4.42  4.84 -3.84  3.39
```

Then we can try two types of ensembles. First, we use bagging's idea to fit each NN model on the bootstrap samples. We fixed the number of hidden units as five and weight decay is zero (*why*?) We see the bagged estimate achieves better performance on the test set than using a single well-chosen NN model.

```
> B <- 50; n <- nrow(data.train)
> pred.boot <- matrix(0,5000,B)
> for (i in 1:B){
   set.seed(i)
   indx <- sample(1:n,size=n,replace=T)
   nnmdl <- nnet(Y~.,data.train[indx,],trace=F,size=5)
   pred.boot[,i] <- predict(nnmdl,data.test,type="raw")
 }
> pred2 <- apply(pred.boot,1,mean)
> tab2 <- table(pred2>=.5,data.test$Y)
> 1-sum(diag(tab2))/sum(tab2)

[1] 0.2778
```

Then we use the random forest's idea to not only use bootstrap samples but also randomly selected variables to build each NN model. We see that by using randomly selected features (5 out of 10 features are chosen to fit each NN model) and the size of hidden units is 3 (compared to 5 in above) we further improved the results!

```
> pred.boot2 <- matrix(0,5000,B)
> for (i in 1:B){
   set.seed(i)
   indx <- sample(1:n,size=n,replace=T)
   indv <- sample(1:10,size=5,replace=F)
   nnmdl <- nnet(Y~.,data.train[indx,c(indv,11)],trace=F,size=3)
   pred.boot2[,i] <- predict(nnmdl,data.test[,indv],type="raw")
 }
> pred3 <- apply(pred.boot2,1,mean)
> tab3 <- table(pred3>=.5,data.test$Y)
> 1-sum(diag(tab3))/sum(tab3)
```

```
[1] 0.2356
```

Reference

Julian J. Faraway and Chris Chatfield (1998) "Time series forecasting with neural networks: a comprehensive study using the airline data" in *Applied Statistician*, **Vol. 47**, 231-250.

Julian J. Faraway (2006) *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models* (1st edition) by Chapman and Hall/CRC.