1. 데이터 소개 및 전처리

1.1. 분석의 목적 및 데이터 소개

송전선은 도시에 전력을 공급하기 위해 수백 마일에 달합니다. 이처럼 거리가 멀어지면 절연체의 결함 혹은 외부물체와의 부딪침으로 인해 부분 방전으로 이어질 가능성이 높습니다. 이러한 부분 방전은 전력선을 천천히 손상 시키므로 수리하지 않은 상태로 두면 결국 정전으로 이어 지거나 화재가 발생합니다. 이러한 부분 방전을 감지하는 솔루션을 개발하면 유지 보수비용을 줄이고 정전을 예방할 수 있습니다.

데이터는 VEB의 ENET Center에서 설계된 미터를 사용하여 얻은 신호로 다음의 링크에서 열린 대회 데이터입니다. 각 신호에는 20밀리 초 이상 걸리는 80만 개의 전력선 전압 측정 값이 포함됩니다. 그리드 자체는 3상 전원 구성표로 작동하며 3상 모두 동시에 측정됩니다. 구성된 데이터의 필드를 자세히 살펴보면 아래와 같이 구성되어 있습니다.

- **metadata_[train/test].csv (114KB/256KB)**
    - id_measurement : 동시에 기록된 트리오 신호의 ID 코드.
    - signal_id : 신호 데이터의 외래 키.
    - phrase : 신호 트리오 내의 위상 ID 코드.
    - target : 결함이 생기면 1, 그렇지 않으면 0.
- **[train/test].csv (3.54GB/8.52GB)**
    - signal : 20밀리 초 이상 걸리는 800,000개의 전압 측정 값

1.2. 데이터 전처리 코드 (전체 코드 삽입)

- 기존의 데이터는 용량이 너무 크고, 정산군과 비정상군이 0.94 : 0.06으로 unbalanced한 문제 이다. 이를 해결하기 위해 정상군을 down sampling해서 일부분만을 사용하도록 한다.
- metadata에 있는 target이 1인 위치와 0인 위치만 뽑아서 train에서 이를 분리한다.
- 마지막으로, 추출한 columns의 순서가 맞도록 sort함수를 이용해서 수정해준다.
- 참고로 metatrain은 signal이 row를 기준으로 하는 반면, train에서는 column을 기준으로 한다.

```
x1 = train[,sample(which(metatrain$target==0), size=100)]
x2 = train[,which(metatrain$target==1)]

x1 = x1[, sort(colnames(x1))]
x2 = x2[, sort(colnames(x2))]
```

1.3. 데이터 전처리 결과 (일부 데이터만 스크린샷 하여 삽입)

- 정상군의 signal 정보

	X105 <int>	X108 <int>	X111 <int>	X122 <int>	X123 <int>	X126 <int>	X128 <int>	X13 <int>	X135 <int>
1	-6	17	11	-12	11	-16	1	-17	14
2	-2	16	11	-10	11	-16	1	-16	13
3	-5	16	10	-12	11	-17	1	-16	10
4	-7	16	11	-13	11	-16	1	-17	12
5	-2	16	10	-11	11	-17	0	-17	15
6	-6	16	11	-15	10	-18	1	-17	14

- 비정상군의 signal 정보

	X201 <int>	X202 <int>	X228 <int>	X229 <int>	X230 <int>	X270 <int>	X271 <int>	X272 <int>	X279 <int>
1	-22	14	4	10	-23	-7	-11	17	-11
2	-23	13	4	10	-24	-5	-9	19	-11
3	-24	11	4	9	-24	-7	-11	18	-12
4	-21	15	3	9	-23	-7	-12	15	-13
5	-25	11	3	9	-23	-7	-10	18	-12
6	-21	15	2	7	-25	-8	-12	16	-10

2. 분석 방법론

2.1. 분석 방법론 설명

분석방법론은 크게 다음의 3가지의 과정으로 이루어진다.

- 노이즈 제거
- 특징추출
- 분류 모델 적용

노이즈 제거 및 특징 추출의 경우 haar 방법과 la8방법을 level1부터 8까지를 진행한다. 이것을 각각 phase 별로 정상군과 비정상군에 적용해서 특징을 추출한다. 이렇게 추출된 정상군과 비정산군의 특징의 평균 차이가 큰 값을 가장 최적의 방법으로 설정하여 가장 최적의 노이즈 제거 방법과 수준을 결정한다. 물론, 모든 신호별로 진행하면 더 좋은 결과를 얻을 수 있지만 시간이 너무 오래 걸리는 관계로 phase별로 진행한다.

위의 방법으로 추출한 특징을 3가지 분류모델인 Random Forest, Neural Network, SVM에 적용한다.

2.2. 분석코드 (전체 코드 삽입)

```
# 1. 노이즈 제거
Denoising = function(x,level,func){

    temp = dwt(x,func,level)

    sigma = sqrt(2*log(length(x)))*sd(temp$d1) ## universal threshold

    ## Hard thresholding rule
    for(i in 1:level){

        temp[[i]][which(abs(temp[[i]])<sigma)] = 0

        }

    value = idwt(temp) ## inverse transform

    return(value)

    }


## Haar wavelet
x1_haar0 = Denoising(x1$X579,4,"haar")
x1_haar1 = Denoising(x1$X415,4,"haar")
x1_haar2 = Denoising(x1$X44,4,"haar")

## Daubechies wavelet(1992)
x1_wavelet0 = Denoising(x1$X579, 4,"la8")
x1_wavelet1 = Denoising(x1$X415,4,"la8")
x1_wavelet2 = Denoising(x1$X44,4,"la8")
```

```
## Haar wavelet
x2_haar0 = Denoising(x2$X270,4,"haar")
x2_haar1 = Denoising(x2$X436,4,"haar")
x2_haar2 = Denoising(x2$X437,4,"haar")

## Daubechies wavelet(1992)
x2_wavelet0 = Denoising(x2$X270,4,"la8")
x2_wavelet1 = Denoising(x2$X436,4,"la8")
x2_wavelet2 = Denoising(x2$X437,4,"la8")

## 고장이 나기전, 후에 hear 및 la8 노이즈 제거 방법을 사용.
par(mfcol=c(3,3), pty="m", mar=c(2,2,2,2))
plot(1:length(x1_haar0),x1$X579,type="l")
plot(1:length(x1_haar0),x1_haar0,type="l")
plot(1:length(x1_haar0),x1_wavelet0,type="l")

plot(1:length(x1_haar0),x1$X415,type="l")
plot(1:length(x1_haar0),x1_haar1,type="l")
plot(1:length(x1_haar0),x1_wavelet1,type="l")

plot(1:length(x1_haar0),x1$X44,type="l")
plot(1:length(x1_haar0),x1_haar2,type="l")
plot(1:length(x1_haar0),x1_wavelet2,type="l")



par(mfcol=c(3,3), pty="m", mar=c(2,2,2,2))
plot(1:length(x1_haar0),x2$X270,type="l")
plot(1:length(x1_haar0),x2_haar0,type="l")
plot(1:length(x1_haar0),x2_wavelet0,type="l")


plot(1:length(x1_haar0),x2$X436,type="l")
plot(1:length(x1_haar0),x2_haar1,type="l")
plot(1:length(x1_haar0),x2_wavelet1,type="l")


plot(1:length(x1_haar0),x2$X437,type="l")
plot(1:length(x1_haar0),x2_haar2,type="l")
plot(1:length(x1_haar0),x2_wavelet2,type="l")



Spectrum = function(x,level,func){

    value = matrix(0,0,nrow=ncol(x),ncol=2)

    for(i in 1:ncol(x)){

        temp = dwt(x[,i],func,level)

        x.axis = 1:level

      y.axis = c()

        for(j in 1:level){
```

```
      y.axis[j] = log2(mean(temp[[j]]^2))

        }

      reg = lm(y.axis ~ x.axis)

      value[i,] = coef(reg)

        }


    return(value)

}


check_level <- function(normal, abnormal, level){
  best_diff = 0
  best_level = 0
  best_denoise = 'haar'
  for(i in 1:level){
    fea1 = Spectrum(normal,i,"haar")
    fea2 = Spectrum(abnormal,i,"haar")
    fea3 = Spectrum(normal,i,"la8")
    fea4 = Spectrum(abnormal,i,"la8")
    # print(abs(mean(fea3[,1]) - mean(fea4[,1])))
    if (abs(mean(fea1[,1]) - mean(fea2[,1])) < abs(mean(fea3[,1]) -
mean(fea4[,1]))){
       diff = abs(mean(fea3[,1]) - mean(fea4[,1]))
       if(diff > best_diff){
         best_diff <- diff
         best_level <- i
         best_denoise <- 'la8'
       }
    }else{
       diff = abs(mean(fea1[,1]) - mean(fea2[,1]))
       if(diff > best_diff){
         best_diff <- diff
         best_level <- i
         best_denoise <- 'haar'
       }
    }

  }
  return(c(best_level, best_denoise))
}


phase0 = c()
phase1 = c()
phase2 = c()
num = 0

for(i in colnames(x1)){
  num = num + 1
  colnum = as.numeric(gsub("[^0-9]", "", i))
  if (colnum %%3 == 0){
    phase0 = append(phase0, num)
```

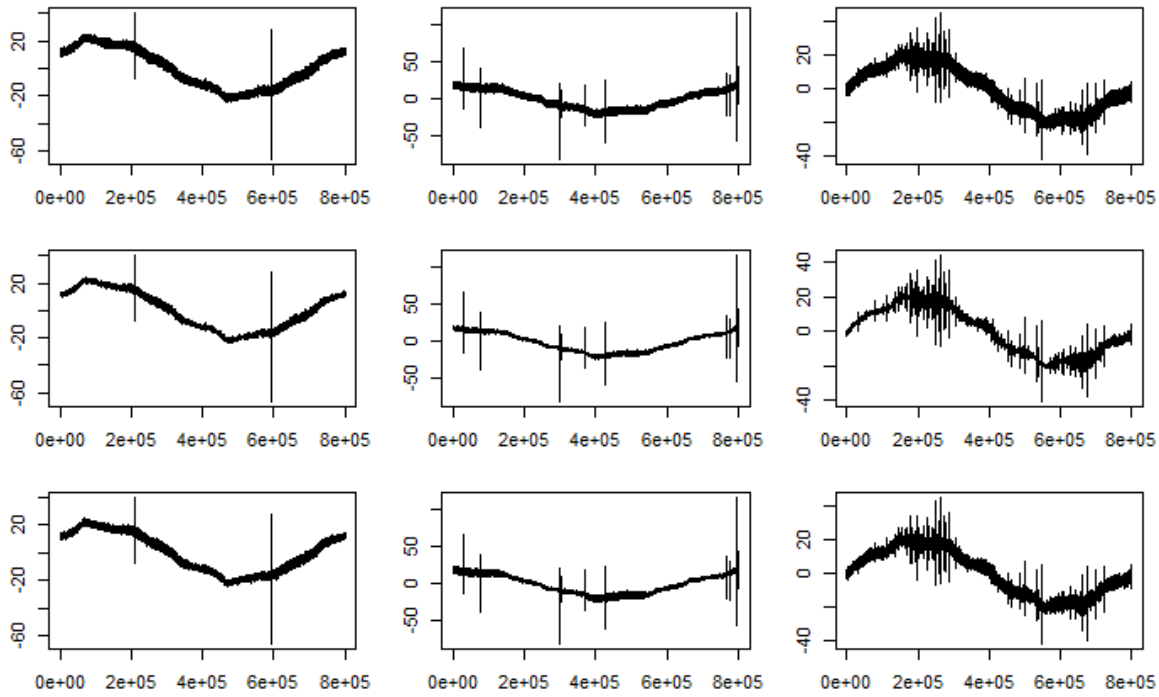
```
  }
  else if (colnum %%3 == 1){
    phase1 = append(phase1, num)
  }
  else {
    phase2 = append(phase2, num)
  }
}

x1_phase0 = x1[,phase0]
x1_phase1 = x1[,phase1]
x1_phase2 = x1[,phase2]

phase0 = c()
phase1 = c()
phase2 = c()
num = 0

for(i in colnames(x2)){
  num = num + 1
  colnum = as.numeric(gsub("[^0-9]", "", i))
  if (colnum %%3 == 0){
    phase0 = append(phase0, num)
  }
  else if (colnum %%3 == 1){
    phase1 = append(phase1, num)
  }
  else {
    phase2 = append(phase2, num)
  }
}

x2_phase0 = x2[,phase0]
x2_phase1 = x2[,phase1]
x2_phase2 = x2[,phase2]


check_level(x1_phase0, x2_phase0, 8)
check_level(x1_phase1, x2_phase1, 8)
check_level(x1_phase2, x2_phase2, 8)


fea1 = Spectrum(x1_phase0,8,"la8")
fea2 = Spectrum(x2_phase0,8,"la8")
fea3 = Spectrum(x1_phase1,8,"la8")
fea4 = Spectrum(x2_phase1,8,"la8")
fea5 = Spectrum(x1_phase2,8,"la8")
fea6 = Spectrum(x2_phase2,8,"la8")

fea1 = data.frame(fea1,"N")
fea2 = data.frame(fea2,"Y")
fea3 = data.frame(fea3,"N")
fea4 = data.frame(fea4,"Y")
fea5 = data.frame(fea5,"N")
fea6 = data.frame(fea6,"Y")

dimnames(fea1)[[2]] = c("coef","slope","Y")
dimnames(fea2)[[2]] = c("coef","slope","Y")
```

```
dimnames(fea3)[[2]] = c("coef","slope","Y")
dimnames(fea4)[[2]] = c("coef","slope","Y")
dimnames(fea5)[[2]] = c("coef","slope","Y")
dimnames(fea6)[[2]] = c("coef","slope","Y")

fea_0 = rbind(fea1,fea2)
fea_1 = rbind(fea3,fea4)
fea_2 = rbind(fea5,fea6)

fea = rbind(fea_0,fea_1)
fea = rbind(fea,fea_2)

wh = sample(1:nrow(fea),round(nrow(fea)*0.8))
train = fea[wh,]
test = fea[-wh,]


library(randomForest)

set.seed(83)
model1 = randomForest(Y ~ coef + slope, data = train,mtry = floor(sqrt(5)),
importance=T, ntree=1000)
plot(model1)


model1 = randomForest(Y ~ coef + slope, data = train, mtry = floor(sqrt(5)),
importance=T, ntree=70)
pre1 = predict(model1, test, type="response")
mat1 = table(test$Y,pre1)
mat1



library(neuralnet)
#model1 = neuralnet(Y ~ coef + slope, data=train,hidden=10)
#plot(model1)

model2 = neuralnet(Y ~ coef + slope, data=train, hidden=c(5, 4),linear.output =
F, stepmax = 1e6)

plot(model2)

# parameter Tuning
best_error = 10
parameter = c()
for(i in 3:5){
  for(j in 3:4){
    #for(l in 0:5){
      model2 = neuralnet(Y ~ coef + slope, data=train, hidden=c(i,
j),linear.output = F, stepmax = 1e6)
      error = model2$result.matrix[,1][[1]]
      if(error < best_error){
        best_error <- error
        parameter <- c(i,j)
      }
    #}
  }
}
```

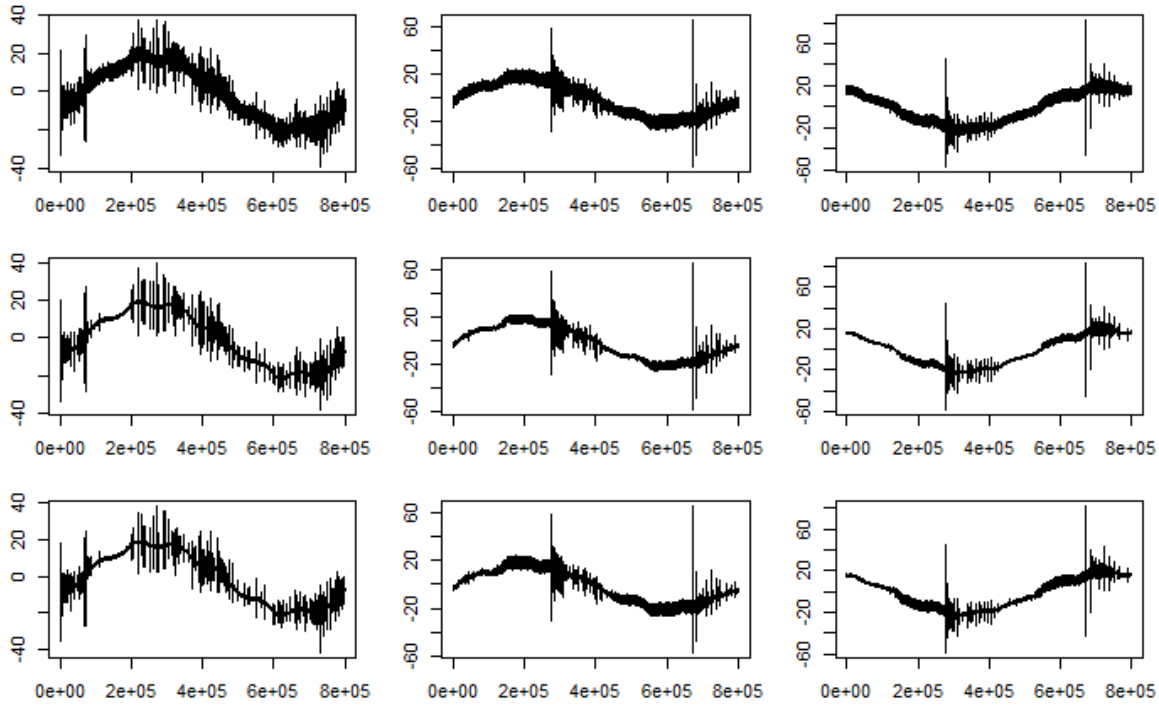
```
print(c(best_error, parameter))



model2 = neuralnet(Y ~ coef + slope, data=train, hidden=parameter,linear.output
= F, stepmax = 1e6)
plot(model2)


pre2 = predict(model2, test, type="response")
pre2 = pre2[,1]
pre2 = ifelse(pre2 < 0.5, 'N','Y')
mat2 = table(test$Y,pre2)
mat2


#install.packages("kernlab")
library(kernlab)
model3 = ksvm(Y ~ coef + slope, data=train, kernel="rbfdot", kpar=list(sigma=2),
C=3)
plot(model3)

pre3 = predict(model3, test, type="response")
mat3 = table(test$Y,pre3)
mat3


confusionMatrix(mat1)
confusionMatrix(mat2)
confusionMatrix(mat3)
```

2.3. 분석결과 및 해석

위에서 설명하였지만, 노이즈 제거는 'haar'과 'la8'방법을 비교하여 사용한다. 먼저 phase별로 정상군과 비정상군에서 어떤 모습을 보이는지 level을 4로 고정하고 살펴본다. 그래프를 살펴보면 왼쪽에서부터 phase 0, 1, 2의 신호이고 위에서부터는 정상군의 신호, haar 방법, la8방법으로 노이즈를 제거했을 때의 결과물이다.



위의 그래프를 통해 확인할 수 있는 점은 phase마다 확연히 구분되는 신호를 가지고, haar 방법과 la8방법을 사용해서 노이즈를 제거할 경우에 뚜렷한 차이를 보이지 않는다는 점이다. 이러한 양상은 비정상군의 신호에서도 동일하게 나타났다.



정상군과 비정상군은 분명 뚜렷한 차이를 보이고 이러한 차이를 수치화 하기 위해 haar과 la8방법으로 노이즈를 제거한 데이터에 선형회귀모델을 적용해 구한 계수를 특징으로 사용할 것이다. 정상군과 비정상군의 특징(계수값)의 평균이 크면 클수록 더 좋은 특징이므로 가장 평균의 차이가 큰 노이즈 제거방법과 수준을 그리드 서치 방법을 이용해서 구한다.

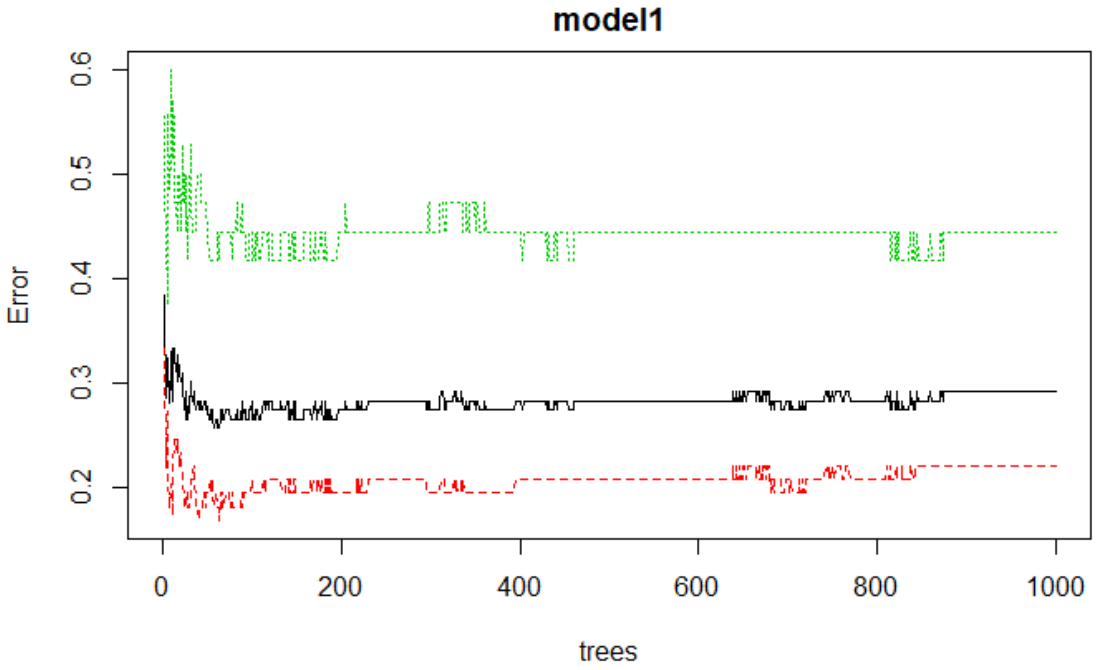
```
> check_level(x1_phase0, x2_phase0, 8)
[1] "8"   "la8"
> check_level(x1_phase1, x2_phase1, 8)
[1] "8"   "la8"
> check_level(x1_phase2, x2_phase2, 8)
[1] "8"   "la8"
```

그리드 서치 결과 phase가 0부터 2까지 모두 정상군과 비정상군의 차이가 가장 크게 나오는 방법은 'la8'방법이 수준8일때 이다.

이렇게 추출한 최적의 특징을 가지고 train과 test를 8:2로 나누어서 RandomForest, NeuralNet, KernelSVM에 적용시켜서 전선이 고장나는지를 탐지시켜본 결과는 아래와 같다.

1. Random Forest

의사결정나무를 여러개를 모아서 Bagging 형태로 앙생블한 Random Forest의 결과는 아래와 같다.여기서 초록색은 비정상군에 대한 error rate이고 빨간색은 정상군에 대한 error rate이고 검정색은 둘의 평균 error rate이다. 검정색 error rate를 보면 trees의 갯수가 70부분에서 최적의 값을 가지고 그 이상으로 갈 수록 Error가 커지다가 점점 수렴하는 형태이다.

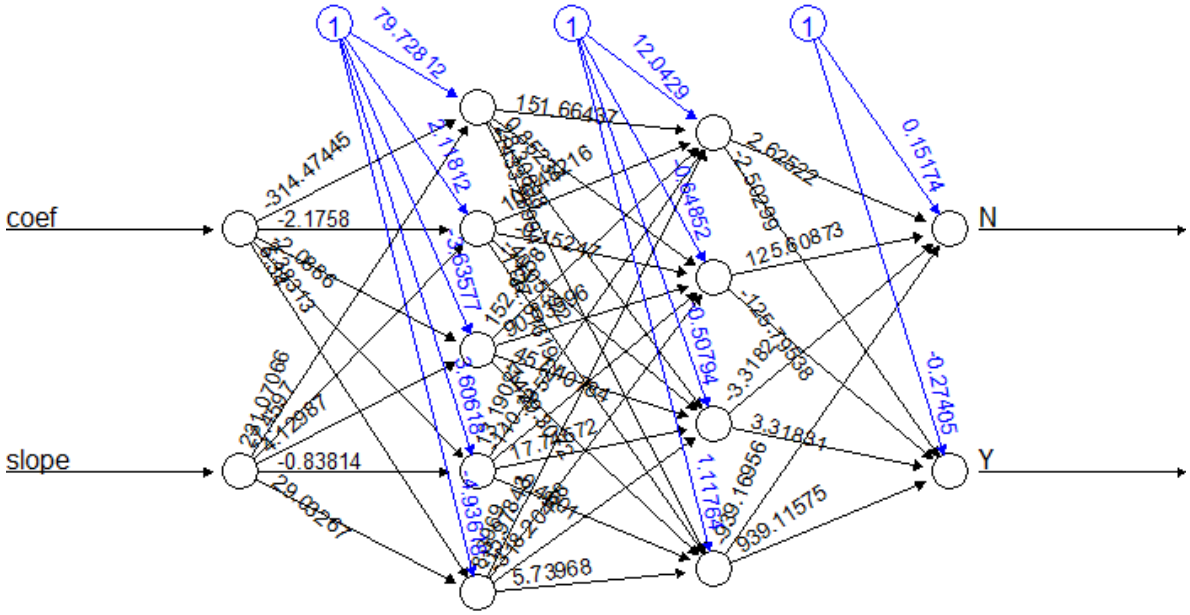


trees수를 80으로 고정시키고 이때 test에 얼마나 잘 적합되었는지 결과를 확인해보면, 아래와 같으며 3개를 제외하고는 모두 맞친 것을 볼 수 있다.

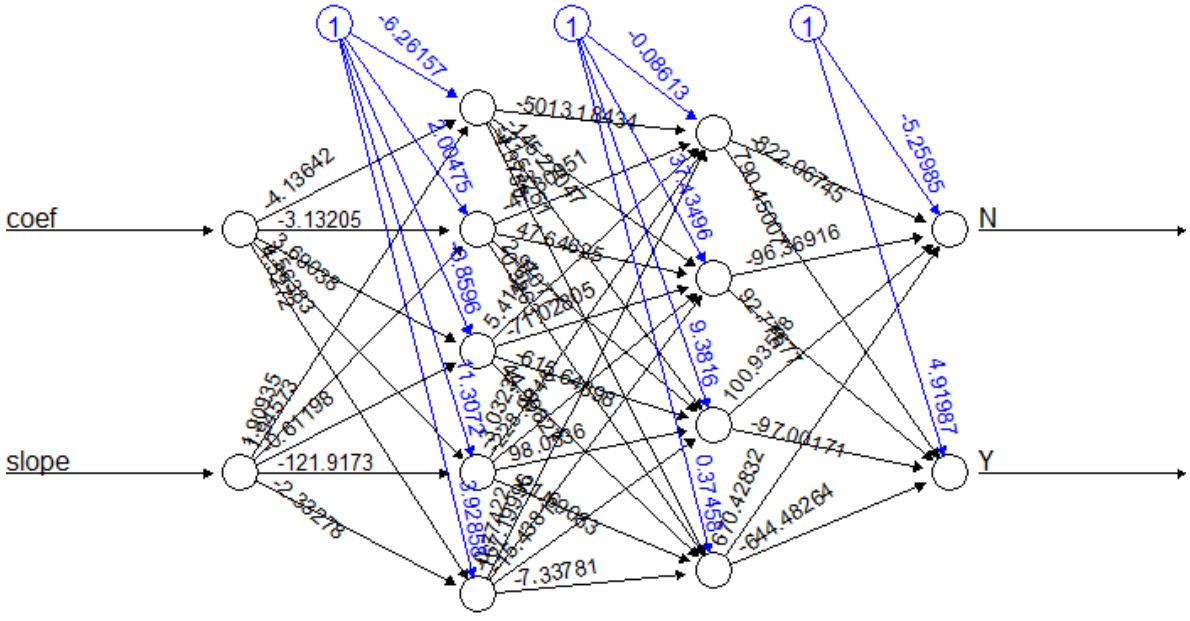
```
   N  Y
N 20  3
Y  0  5
```

2. Neural Network

인공신경망기법은 최근에 각광 받은 기술로서 비선형형태의 데이터를 적합시킬 때 좋은 방법이다. 모델의 학습결과는 아래와 같으며 Error만 보면 Random Forest에 비해 낮은 것을 확인할 수 있다.



모델의 성능을 높이기위해서 Hyperparameter을 그리드서치방법을 이용해서 튜닝을 하면, error을 3.79까지 줄일 수 있다.

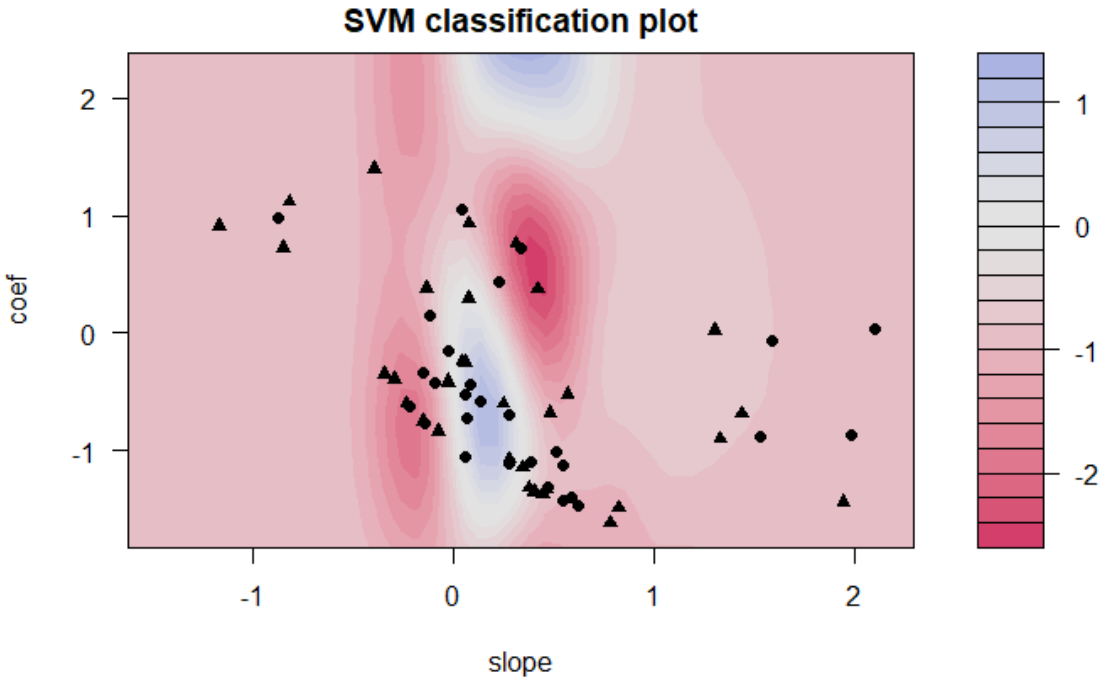



```
  pre2
    N  Y
N   6 17
Y   4  1
```

하지만 Train에 오버피팅되어서 실제 결과는 별로 안좋은 것을 확인할 수 있다.

3. SVM

SVM은 Support vector들이라는 vector들과의 거리를 계산해서 정상군과 비정상군을 구분하는 경계를 만드는 기법이다.




```
   pre3
     N  Y
N 21  2
Y  0  5
```

SVM의 결과를 보면 이제까지 모델중에서 가장 좋은 결과를 보이고 있다. 이제까지의 결과를 종합하면, 아래의 테이블을 얻을 수 있다.

	Random Forest	**Neural Network**	**SVM**
Accuracy	0.8929	0.25	0.9286
Sensitivity	1	0.6	1
Specificity	0.6250	0.055	0.7143