Comparação de técnicas de regressão em uma estratégia simples para economizar energia em redes de sensores sem fio

Antonio Alex de Souza
aasouzaconsult@gmail.com

Dezembro 2017

1 Introdução

Este trabalho tem como objetivo comparar de técnicas de regressão em uma estratégia simples para economizar energia em redes de sensores sem fio. Daremos uma breve descrição do ambiente e recursos utilizados e em seguida, a implementação das técnicas. O código produzido pode ser encontrado em https://github.com/aasouzaconsult/SensoresIntel/blob/master/SensoresIntel.r

2 Ambiente e Recursos

2.1 R

Para produção deste trabalho, foi utilizado o R (https://cran.r-project.org).

2.2 Dataset dos Sensores

Os dados a serem utilizados foram coletados no Laboratório de Pesquisa da Intel. Estão dispostos conforme podemos observar na figura 1:

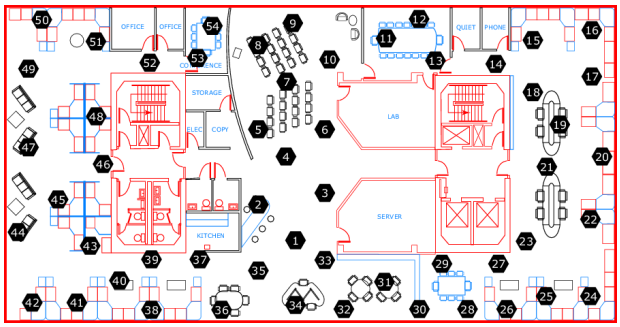

Figura 1: Disposição dos nós sensores.

Usaremos os dados pré-processados, disponíveis em http://www.ulb.ac.be/di/labo/code/PCAgExpe.zip. Nele foram excluídos dois sensores considerados defeituosos dentro do período em que a amostra foi coletada e são considerados 14400 medições divididos em época, que serão representados por linha. Estes sensores são os de número 5 e 15. Para este trabalho, só utilizaremos os dados de temperatura.

Para conversão dos dados em valores numéricos, a seguinte função pode ser utilizada, para ambos os arquivos (mote_locs.txt (posicionamento sensores) e subsfin.txt (temperaturas)):

```
converte = function(dados) {
  c<-NULL
  for (i in 1:ncol(dados))
    c<-cbind(c,as.numeric(dados[,i]))

  return (c)
}
```

3 Técnicas utilizadas

Serão aplicadas cinco técnicas, cada uma detalhadamente explicada logo mais abaixo

3.1 Kernel Gaussiano (item a)

Programar uma função de interpolação com kernel gaussiano para estimar a superfície de temperatura de uma época qualquer dos dados. Calcular o valor da temperatura em 50 pontos de teste localizados aleatoriamente na área sensoriada. Mostrar em um gráfico a superfície de interpolação, localizando os pontos dos sensores e os 50 pontos de teste. Para aprender o parâmetro abertura do kernel gaussiano, utilizar as primeiras 300 épocas dos dados. Os valores de temperatura estimados por essa função de interpolação serão considerados os valores de referência (ground truth) para os pontos de teste. Saída: códigos dos procedimentos, a função de interpolação e os gráficos do campo sensor para algumas épocas escolhidas.

Para este item será utilizada uma função de interpolação com Kernel Gaussiano para estimar a superfície de temperatura de uma época qualquer dos dados.

O modelo Nadaraya-Watson será usado para cálculo da regressão, abaixo a fórmula utilizada:

$$y(x) = \frac{\sum_n g(x - x_n)t_n}{\sum_m g(x - x_m)} \tag{1}$$

E a função g, baseada no método de Kernel Gaussiano:

$$g(x - x_n) = \left(\frac{1}{2\pi\sigma^2}\right) \exp^{\left(-\frac{(x-x_n)^2}{2\sigma^2}\right)} \tag{2}$$

Abaixo descrevemos passo a passos todas as etapas do algoritmo, basicamente serão dispostos em 5 principais passos, são eles:

1. Importando os dados utilizando a função de conversão para número:

```
dados = converte(read.table("C:/temp/subsfin.txt")) #temperaturas
locs  = converte(read.table("C:/temp/mote_locs.txt")[,2:3]) #sensores
locs  = locs[c(-5,-15),] #sensores removidos -queimados
```

2. Função para calcular o parâmetro de abertura (utilizando os 300 primeiras linhas e calculando o desvio padrão)

```
estima_sigma = function(dados){
  sigma = c() # Vetor Sigma
  g_t = dados[1:300,]
  for(i in 1:ncol(dados)){
    sigma[i] = sd(g_t[,i]) #Desvio Padrao para cada um dos 52 sensores
  }
  return (sigma)
}
```

3. Função para gerar pontos para validação na aréa do laboratório

```
gera_pontos = function(n){
  # Cria matriz de 0 com N linhas e 2 colunas
  pontos = matrix(0, nrow = n, ncol=2)

  # Gera n pontos entre o minimo e o maximo de x(locs[,1]) e de y (locs[,2])
  pontos[,1] = sample(min(locs[,1]):max(locs[,1]), n, replace = TRUE)
  pontos[,2] = sample(min(locs[,2]):max(locs[,2]), n, replace = TRUE)
  return (pontos)
}
```

4. Função Kernel (implementação da fórmula (2))

```
kernel_gauss = function(locs , x, sigma){
  m_x = t(replicate(52, x)) # x e cada um dos pontos gerados

  # Distancia Euclidiana
  dst = (m_x - locs)^2 # Distancia de cada ponto gerado para com os originais
  dst = dst[,1]+dst[,2]

  pt1 = 1/(2*pi*(sigma^2))
  pt2 = exp(-(dst/(2*(sigma^2))))
  result = pt1*pt2

  return(result)
}
```

5. Execução do Cálculo de Regressão

   A seguir, passos da execução:

```
# Epoca Testada
epoca = 301

# Desvio padrao de cada um dos 52 sensores
sigma = estima_sigma(dados)

# 52 temperaturas da Epoca Testada
dados_epoca = dados[epoca,]

# Gera 50 pontos na area
pontos = gera_pontos(50)
```

- Execução da função: kernel_gauss (Fórmula 1 - Nadaraya-Watson) e armazenamos os resultados (regressão)

```
1
2 # Chamada da funcao para estimar as temperaturas nos 50 pontos gerados (
      ground truth)
3 result = c() # cria um vetor de resultados
4 for(i in 1: nrow(pontos)){
5   # (Formula 2 - Kernel Gaussiano)
6   k = kernel_gauss(locs, pontos[i,], sigma) # k s o 52 numeros
7
8   # (Formula 1 - Nadaraya-Watson)
9   result[i] = (k%*%dados_epoca)/sum(k)
10 }
```

- Concatenando dados do dataset (originais) com os resultados (gerados). Foi criado uma matriz 102X4 para armazenar esses valores, onde a coluna 1 representa os valores de x, a coluna 2 os valores de y, a coluna 3 as temperaturas e a coluna 4 informa se são os dados originais (1) ou os gerados(2).

```
1 x  = c(locs[,1], pontos[,1]) # x (102 pontos - 52 originais e 50 gerados)
2 y  = c(locs[,2], pontos[,2]) # y (102 pontos - 52 originais e 50 gerados)
3 tn = c(dados_epoca, result)  # Temper.  (102 - 52 originais e 50 gerados)
4
5 coord = matrix(0, ncol = 4, nrow=102)
6 coord[1:52,1]   = locs[,1]    # x dos dados originais
7 coord[53:102,1] = pontos[,1]  # x dos 50 pontos - Previstos
8 coord[1:52,2]   = locs[,2]    # y dos dados originais
9 coord[53:102,2] = pontos[,2]  # y dos 50 pontos - Previstos
10 coord[1:52,3]   = dados_epoca # temperaturas originais
11 coord[53:102,3] = result      # temperaturas previstas
12 coord[1:52,4]   = 1           # Originais
13 coord[53:102,4] = 2           # 50 pontos - Previstos
```

- Montando a superfície, desenhando os pontos (originais: pretos e gerados: vermelhos) e plotando a superfície.

```
1 library(akima)
2 library(rgl)
3 shape = interp(locs[,1], locs[,2], dados_epoca,
4                xo=seq(min(locs[,1]), max(locs[,1]), length=600),
5                yo=seq(min(locs[,2]), max(locs[,2]), length=600))
6
7 # Visao 3D - Dados Originais (invertido y por z)
8 plot3d(coord[1:52,1], coord[1:52,3], coord[1:52,2], ylim=c(0,40),
9        type = "s",
10       col = "black",
11       size = 1,
12       xlab = "x",
13       ylab = "z",
14       zlab = "y")
15
16 # Desenhando os pontos gerados (invertido y por z)
17 plot3d(coord[53:102,1], coord[53:102,3], coord[53:102,2], ylim=c(0,40),
18       type = "s",
19       col = "red",
20       size = 1,
21       xlab = "x",
22       ylab = "z",
23       zlab = "y")
24
```

```
25 # Plotando a superficie
26 rgl.surface(shape$x,shape$y,shape$z, color = "orange", alpha=c(0.5))
```

Resultados

Na figura abaixo podemos ver uma visualização da disposição dos dados para uma época (Época: 301)

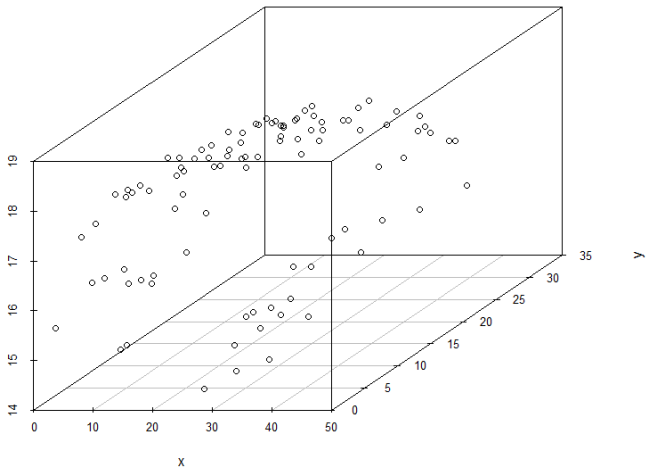

Figura 2: Resultado - Disposição dos dados

Uma visualização 3D (função: plot3D e para gerar a superfície: rgl.surface) da disposição dos dados para uma época (Época: 301) - SURFACE

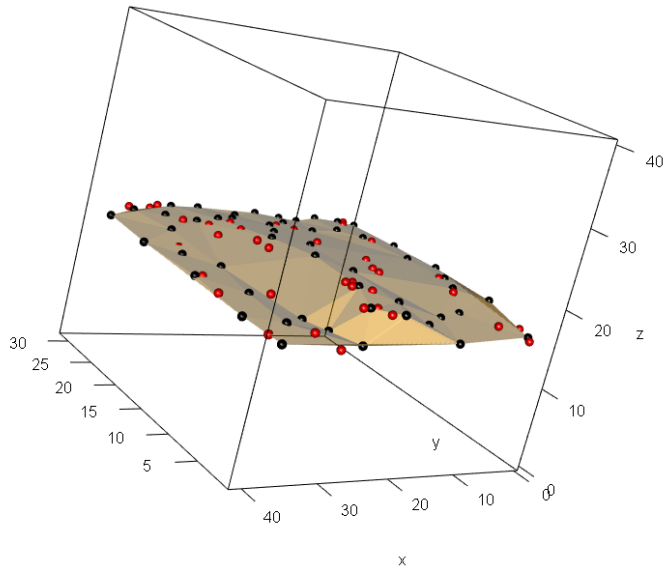

Figura 3: Resultado - Disposição dos dados - SURFACE

3.2 Decisão Descentralizada (item b)

A estratégia (decisão descentralizada) para economizar energia consiste em cada nó sensor decidir autonomamente transmitir ou não transmitir sua medição de cada época com probabilidade p. Nesse caso, em cada época o nó central conhecerá os valores reais da temperatura apenas nos locais dos nós que transmitiram. Para estimar a temperatura nos pontos de teste ele utiliza os últimos valores disponíveis ou estimados nos pontos dos nós sensores. Todos os nós sensores são programados para transmitirem as primeiras 5 épocas. Assim, os dados dessas épocas estarão disponíveis no nó central.
> Estimar os valores de temperatura nos pontos de teste utilizando regressão linear com regularização. Ajustar experimentalmente o parâmetro de regularização. Calcular, sobre todo o dataset: o NRMSE (raiz do erro quadrático médio normalizado) de épocas, os NRMSEs máximos e mínimo, e reter o id e os valores das 10 épocas com NRMSEs máximos e 10 épocas com NRMSEs mínimos. Fazer isso para p= 0.1:0.1:0.9. Saída: Gráfico e tabela com NRMSEs, médio, máximo e mínimo em função de p.

Abaixo descrevemos passo a passos todas as etapas do algoritmo (alguns passos serão reaproveitado dos anteriores), basicamente serão dispostos em 4 principais passos, são eles:

1. Função para calcular a probabilidade (probabilidade de quem vai ou não transmitir)

```
trans_p = function(p, pontos){
  #matriz 14400 x 50 colunas(pontos gerados)
  result = matrix(0, nrow = nrow(dados), ncol = nrow(pontos))
  for(i in 1:nrow(dados)){
    for (j in 1:nrow(pontos)) {
      if(runif(1)<p){ # Numero aleatorio e menor que a probabilidade de
    entrada
        d_tr[j] = dados[i,j]
      }
    }

    #print(length(i_tr))

    result[i,] = reg_lin(d_tr, dados[i,], pontos)
    #print(i_tr)
  }
  return(result)
}
```

2. Função da Regressão Linear

   O cálculo da regressão foi baseado no modelo linear com regularização, como mostrado abaixo, onde lambda é setado como 0.1:

$$w = (X^T X + \lambda I)^{-1} X^T t \tag{3}$$

   Enquanto o cálculo da estimação dos novos valores são obtidos por:

$$y = X\beta + \varepsilon \tag{4}$$

Implementação das fórmulas acima:

```
# Funcao da Regressao Linear
reg_lin = function(i_tr, dados_epoca, nlocs){
    result = array(0, dim = nrow(nlocs)) # array de 0 de nrow(nlocs) + 50 posicoes
  X = matrix(1, ncol = 3, nrow = nrow(locs)) # Matriz de 1 s  (nrow(nlocs) linhas e 3 colunas)
   #X[,1]    o bias
  X[,2] = locs[,1]
  X[,3] = locs[,2]
    #...
    x = X
  I = diag(ncol(X))
  lambda = 0.1
  # Formula 3 (modelo linear com regularizacao)
  v = solve(t(x)%*%x + lambda*I)%*%t(x)%*%d_tr

  for(i in 1:length(result)){
    # y = XB + e - F rmula 4 (calculo da estimacao)
    result[i] = (v[1,1] + v[2,1]*nlocs[i,1] + v[3,1]*nlocs[i,2])
  }
  return (result)
}
```

3. Função para calcular os NRMSEs (Máximo, Médio e Mínimo)

```
# NRMSE - Erro maximo
erromax = function(result, dados){
  re2_max_ep = c()
  for(i in 1:nrow(result)){
    a = (dados[i,] - result[i,])^2
    re2_max_ep[i] = max(sqrt(a))
  }
  return (re2_max_ep)
}

# NRMSE - Erro medio
erromedio = function(result, dados){
  rmse_epoca = c()
  re2_min_ep = c()
  re2_max_ep = c()
  for(i in 1:nrow(result)){
    a = (dados[i,] - result[i,])^2
    rmse_epoca[i] = sqrt(sum(a))/ncol(result)
  }
  return (rmse_epoca)
}

# NRMSE - Erro minimo
erromin = function(result, dados){
  rmse_epoca = c()
  re2_min_ep = c()
  re2_max_ep = c()
  for(i in 1:nrow(result)){
    a = (dados[i,] - result[i,])^2
    re2_min_ep[i] = min(sqrt(a))
  }
  return (re2_min_ep)
}
```

4. Função para reter os 10 NRMSEs (Máximo e Mínimo)

```
# Funcao para reter os 10 NRMSEs maximos
max10 = function(re2_max_ep){
  res2 = re2_max_ep

  ind = c()
  for(i in 1:10){
    a = which.max(res2) # indice do maior valor
    ind[i] = a
    res2[a] = 0
  }
  return(ind)
}

# Funcao para reter os 10 NRMSEs minimos
min10 = function(re2_min_ep){

  res2 = re2_min_ep

  ind = c()
  for(i in 1:10){
    #a = which(mm == min(re2_min_ep), arr.ind = TRUE)
    a = which.min(res2)
    ind[i] = a
    res2[a] = 1000
  }
  return(ind)
}
```

Resultados

Nas figuras abaixo podemos verificar os erros máximos, médios e mínimos respectivamente de cada uma das probabilidades (Probabilidade: 0.1 - Azul, 0.2 - Amarelo, 0.3 - Preto, 0.4 - Vermelho, 0.5 - Verde, 0.6 - Rosa, 0.7 - Laranja, 0.8 - Cinza e 0.9 - Marrom)

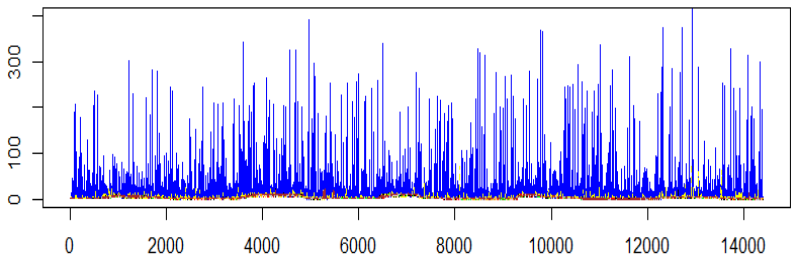

Figura 4: Resultado - Erros Máximos

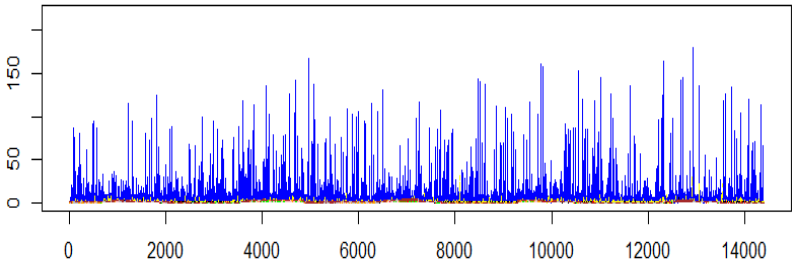

Figura 5: Resultado - Erros Médios

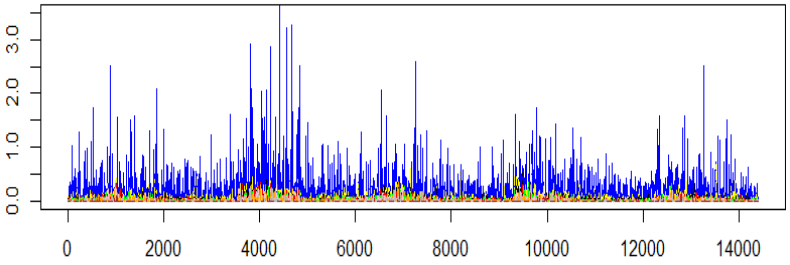

Figura 6: Resultado - Erros Mínimos

Em seguida, mostrando a Tabela de Resultados (Máximo, Média e Mínimo)

SensoresIntel.R | max_matrix | med_matrix | min_matrix

Filter

1	12943	4966	12717	12332	9783	9824	3601	6500	11029	8473
2	12910	8102	13535	13068	7363	12786	7149	11009	5728	3476
3	4259	5292	5293	5289	5290	5291	5294	5303	5466	5257
4	5289	5293	5290	5291	5292	5303	4259	5276	5294	5472
5	5289	5291	5292	5294	5290	5293	5303	4259	5276	5277
6	5289	5292	5290	5291	5293	5303	4259	5276	5277	5294
7	4259	5289	5292	5290	5291	5293	5294	5466	5276	5277
8	5289	5292	5290	5291	5293	5466	5303	4259	5276	5277
9	4259	5289	5290	5291	5293	5294	5292	5303	5276	5267

Figura 7: Tabela de Resultados - Máximo

SensoresIntel.R | max_matrix | med_matrix | min_matrix

Filter

1	12943	4966	12332	9783	9824	10560	12717	11029	8473	12687
2	8102	12910	13535	13068	7363	7149	2388	5728	12786	12388
3	12648	4841	1504	5652	7121	9422	4437	3866	8081	6618
4	7028	3988	6990	7062	7165	4259	7114	7112	5472	6989
5	7027	7024	6893	7058	7043	7109	7215	7151	7007	6961
6	7040	7096	3979	7079	6955	7043	7019	7092	7026	6999
7	7089	7148	7122	7029	7039	7027	7093	7114	6913	6994
8	7145	7025	6976	7057	7136	7182	4005	7142	7058	7223
9	7065	7146	7030	7121	7142	7107	7092	7037	7168	7033

Figura 8: Tabela de Resultados - Médio

SensoresIntel.R | max_matrix | med_matrix | min_matrix

Filter

1	4414	4675	4567	3796	4220	7243	13281	4830	870	1835
2	9679	13517	7149	8182	4617	2678	9316	3850	1452	9367
3	6545	9494	3823	4147	3797	13089	6869	9358	12648	9630
4	3632	4008	4220	9474	9503	1004	3901	4363	6914	3968
5	9614	3944	989	12607	9509	3829	4128	12585	4124	7012
6	9659	12554	4617	3796	3555	3803	3979	972	4012	998
7	6899	9668	4009	3794	6877	9649	4235	3804	6589	6822
8	9554	9661	9595	6827	3806	3803	7136	4270	4327	6810
9	4208	9645	506	6920	1011	4700	12642	643	9962	4168

Figura 9: Tabela de Resultados - Mínimo

Agora, mostramos os erros médios (Máximo, Médio e Mínimo) e em seguida os MAX dos erros máximos, MIN dos erros mínimos e a MED dos erros médios.

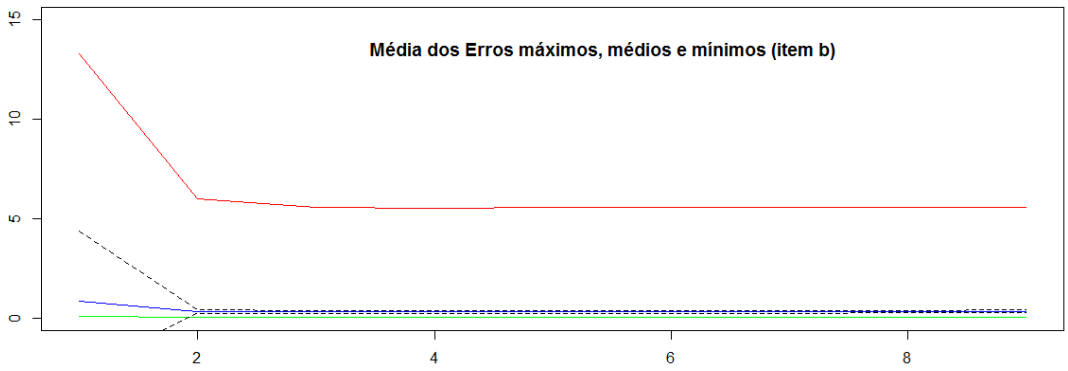

Figura 10: Erros médios

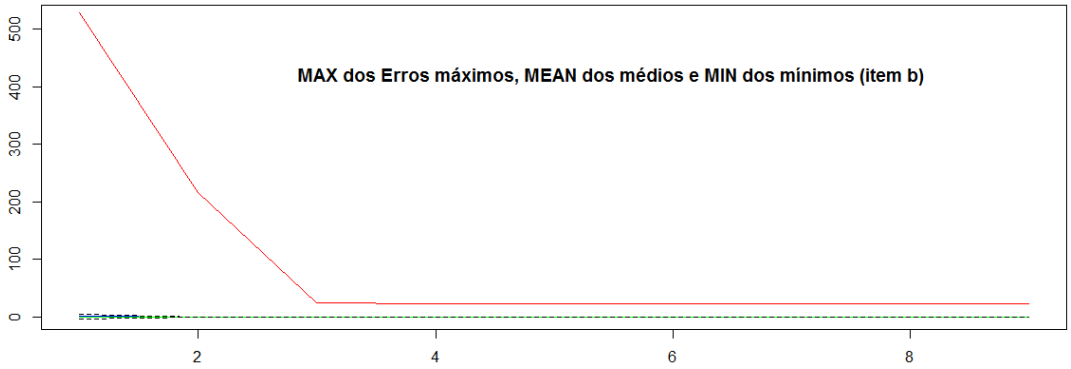

Figura 11: MAX dos erros máximos, MED dos médios e MIN dos mínimos

3.3 Regressão Kernel Gaussiano (Nadaraya-Watson) (item c)

Repetir o item (b) utilizando regressão kernel gaussiano (Nadaraya-Watson).

Abaixo descrevemos passo a passos todas as etapas do algoritmo (alguns passos serão reaproveitado dos anteriores), basicamente serão dispostos em 4 principais passos, sendo o ultimo, a execução... são eles:

1. Função Kernel Gaussiano (Fórmula: 2 - Adaptado ao item c)

```
# Funcao Kernel (Formula 2) - para o Nadaraya Watson
kernel_gauss_nw = function(i_tr , sigma, ponto){
   #Ao inves de 52x - Agora replica pelo numero de sensores escolhidos
  m_x = t(replicate(length(i_tr), ponto))
  dst = (m_x - locs)^2
  dst = sqrt(dst[,1]+dst[,2])

  pt1 = 1/(2*pi*(sigma^2))
  pt2 = exp(-(dst/(2*(sigma^2))))
  result = pt1*pt2

  return(result)
}
```

2. Função Nadaraya Watson - Fórmula: 1

```
nadaraya = function(i_tr, dados_epoca, sigma, ponto){
  k = kernel_gauss_nw(i_tr, sigma, ponto)
  return( (k%*%i_tr)/sum(k) ) # saida = 1 temperatura ("ponto a ponto")
}
```

3. Função (Nadaraya) para calcular a probabilidade (probabilidade de quem vai ou não transmitir)

```
# Dados da primeira epoca
d_tr = dados[1,]

trans_p_nw = function(p, pontos){
  result = matrix(0, nrow = nrow(dados), ncol = nrow(pontos))
  for(i in 1:nrow(dados)){
```

```
    for (j in 1:nrow(pontos)) { # De cada epoca a probabilidade de cada ponto
      if(runif(1)<p){
        d_tr[j] = dados[i,j]
      }
    }
    for(d in 1:nrow(pontos)){
      result[i,d] = nadaraya(d_tr, dados[i,], sigma, pontos[d,]) # saida = 1
    temperatura
    }
  }
  return(result)
}
```

4. Passos da execução:

```
sigma  = estima_sigma(dados)
pontos = gera_pontos(50)

# Chamada da funcao (Formulas: 1 e 2) para estimar as temperaturas de cada
    epoca nos 50 pontos gerados (ground truth)
for(i in 1:nrow(resultT2)){
  for(j in 1:ncol(resultT2)){
    k = kernel_gauss(locs, pontos[j,], sigma)
    resultT2[i,j] = (k%*%dados[i,])/sum(k)
  }
  #print(i)
}

# Monta a regressao kernel gaussiano com probabilidade de 0.1 a 0.9 para as
    50 temperaturas para cada epoca
# Nadaraya Watson
rs1 = trans_p_nw(0.1,pontos)
...
rs9 = trans_p_nw(0.9,pontos)
```

5. Resultados:

   Agora, mostramos os erros médios (Máximo, Médio e Mínimo) e em seguida os MAX dos erros máximos, MIN dos erros mínimos e a MED dos erros médios.

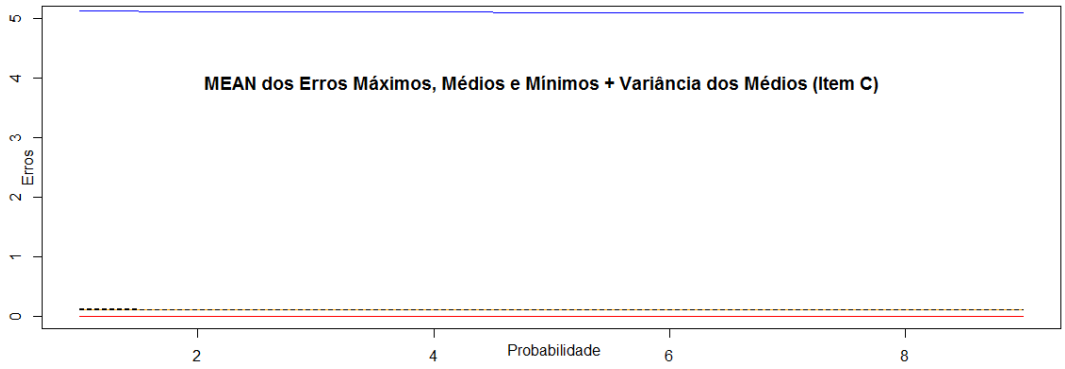

Figura 12: Erros médios

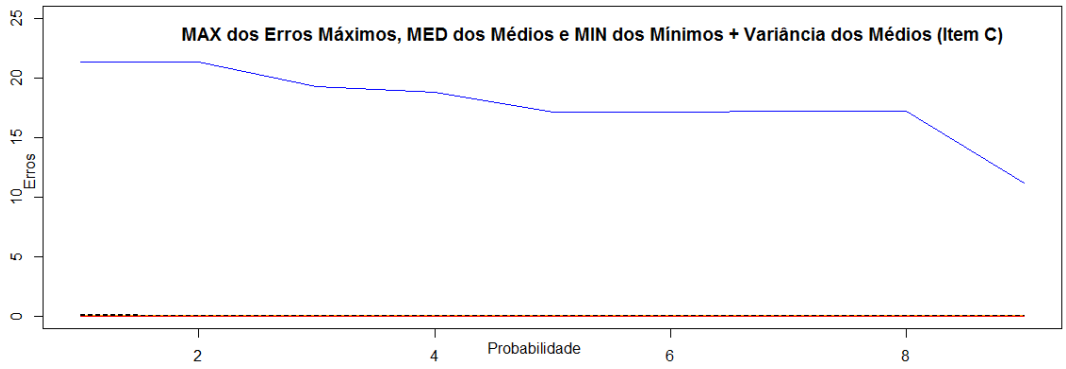

Figura 13: MAX dos erros máximos, MED dos médios e MIN dos mínimos

3.4 Regressão com RBFNN (item d)

Repetir o item (b) utilizando regressão com RBFNN.

Abaixo descrevemos passo a passos todas as etapas do algoritmo (alguns passos serão reaproveitado dos anteriores), basicamente serão dispostos em 4 principais passos, são eles:

1. Função para calcular a probabilidade (probabilidade de quem vai ou não transmitir). Quanto maior a probabilidade, maior o numero de sensores que irão transmitir ("melhor")

```
trans_p = function(p, pontos){

  d_tr = dados[1,] # Primeira linha das temperaturas (base)

  result = matrix(0, nrow = nrow(dados), ncol = nrow(pontos))
  #matriz 14400 x 50 colunas(pontos gerados)

  # Treinando o y
  ytreino = dados * 0

  ytreino[1,] = d_tr

  for(i in 2:nrow(dados)){
    for (j in 1:nrow(locs)) {
      if(runif(1)<p){                  # Se o ponto for escolhido (sera
    transmitido)
        d_tr[j] = dados[i,j]           # Atualiza os dados base (d_tr)
      }
    }
    ytreino[i,] = d_tr
  }

  k = kmeans(locs, 10)                 # K-Means, com k = 10

  w = treino_rbf(ytreino, 10, k)       # Treinando

  a = teste_rbf(pontos, k, 10, w)      # Testando com base nos pontos gerados

  return(a)
}
```

2. Função de teste da RBF

```
teste_rbf = function(pontos, k, n_neuronios, w){
  H = matrix(1, nrow=nrow(pontos), ncol = n_neuronios+1) # 50X11 - 1 Bias
  means    = k$centers
  clusters = k$cluster
  v = array(0, c(2,2, n_neuronios))
  for(i in 1:n_neuronios){
    group = which(clusters %in% i) # Unir os do mesmo grupo
    v[,,i] = var(locs[group,])     # variancia do grupo
  }

  for(i in 1:nrow(pontos)){
    for(j in 1:n_neuronios){
      sigma = diag(1,2)
      diag(sigma) = diag(v[,,j])
      H[i,j+1] = exp(-(t(pontos[i,]-means[j,]) %*% solve(sigma) %*% (pontos[i
    ,]-means[j,])))
    }
  }
  return(t(H%*%w))
}
```

3. Função de treinamento da RBF

```
treino_rbf = function(y, n_neuronios, k){
  H = matrix(1, nrow=nrow(locs), ncol = n_neuronios+1) # Matriz de 1 - 52x11
    (1 - Bias)
  means    = k$centers
  clusters = k$cluster

  # Variancia dos grupos
  v = array(0, c(2,2, n_neuronios))
  for(i in 1:n_neuronios){
    group = which(clusters %in% i)
    v[,,i] = var(locs[group,])
  }

  # H
  for(i in 1:nrow(locs)){
    for(j in 1:n_neuronios){
      sigma = diag(1,2)
      diag(sigma) = diag(v[,,j])
      H[i,j+1] = exp(-(t(locs[i,]-means[j,]) %*% solve(sigma) %*% (locs[i,]-
    means[j,])))
    }
  }
  # W
  return(MASS::ginv(t(H)%*%H)%*%t(H)%*%t(y))
}
```

4. Execução:

```
sigma  = estima_sigma(dados)
pontos = gera_pontos(50)

# Funcao Kernel (Formula 2)
kernel_gauss_gt = function(locs , x, sigma){
  # transposto da repeticao de cada um dos pontos gerados (Matriz - 52X2)
  m_x = t(replicate(52, x))   # x - cada um dos pontos gerados

```

```
  # Distancia Euclidiana (eleva ao quadrado e depois tira a raiz - tirar os
    negativos)
  dst = (m_x - locs)^2          # Distancia de cada ponto gerado para os
    originais (52)
  dst = sqrt(dst[,1]+dst[,2]) # Raiz quadrada

  pt1 = 1/(2*pi*(sigma^2))
  pt2 = exp(-(dst/(2*(sigma^2))))
  result = pt1*pt2

  return(result)
}

# Chamada da funcao Kernel - para estimar as temperaturas nos 50 pontos
    gerados (ground truth)
resultT = matrix(0, nrow = nrow(dados), ncol = nrow(pontos)) # 14400 X 50
for(i in 1:nrow(resultT)){
  for(j in 1:ncol(resultT)){
    k = kernel_gauss_gt(locs, pontos[j,], sigma)  # Distancia Euclidiana
    resultT[i,j] = (k%*%dados[i,])/sum(k)
  }
}

# Monta a RBF com probabilidade de 0.1 a 0.9 para as 50 temperaturas para
    cada Epoca
rs1 = trans_p(0.1,pontos)
rs2 = trans_p(0.2,pontos)
rs3 = trans_p(0.3,pontos)
rs4 = trans_p(0.4,pontos)
rs5 = trans_p(0.5,pontos)
rs6 = trans_p(0.6,pontos)
rs7 = trans_p(0.7,pontos)
rs8 = trans_p(0.8,pontos)
rs9 = trans_p(0.9,pontos)
```

5. Resultados:

   Agora, mostraremos os MAX dos erros máximos, MIN dos erros mínimos e a MED e VAR dos erros médios.

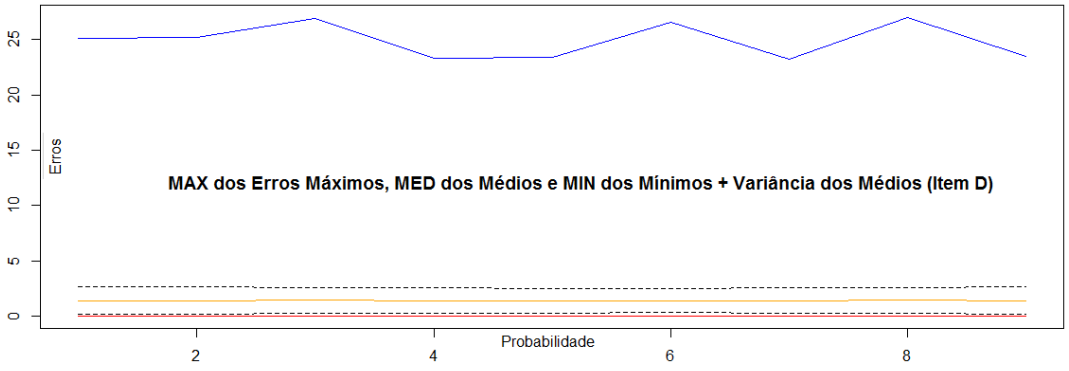

Figura 14: Com Outliers

Observamos valores altos nos erros máximos, e ao analisarmos encontramos alguns outliers, tratamos esses outliers pegando um valor proximo ao da vizinhança, apenas para teste e

obtivemos os seguintes resultados.

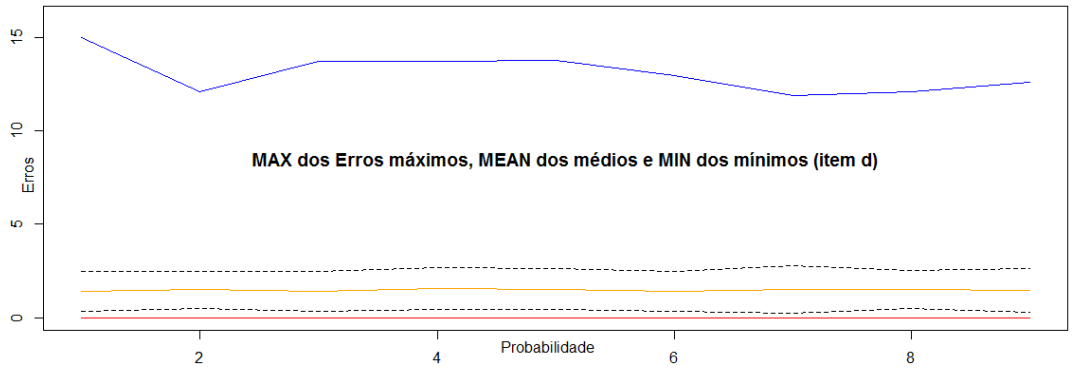

Figura 15: Sem Outliers

3.5 Regressão - Processo Gaussiano (item e)

Repetir o item (b) utilizando regressão processo gaussiano, com kernel de covariância exponencial quadrática.

4 Resultados

A aplicação dos diversos modelos apresentaram pouca variação, observamos também alguns outliers que fazem alguns resultados se tornarem mais altos.