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:

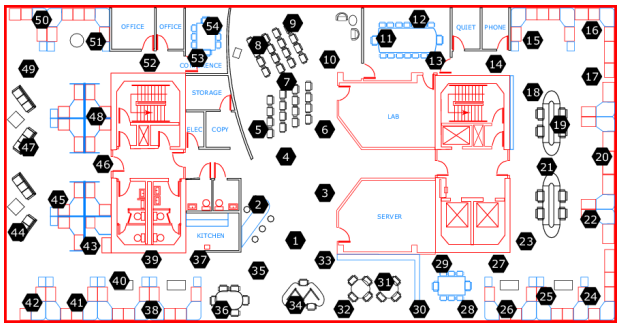

Figura 1: Disposição dos nós sensores.

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)

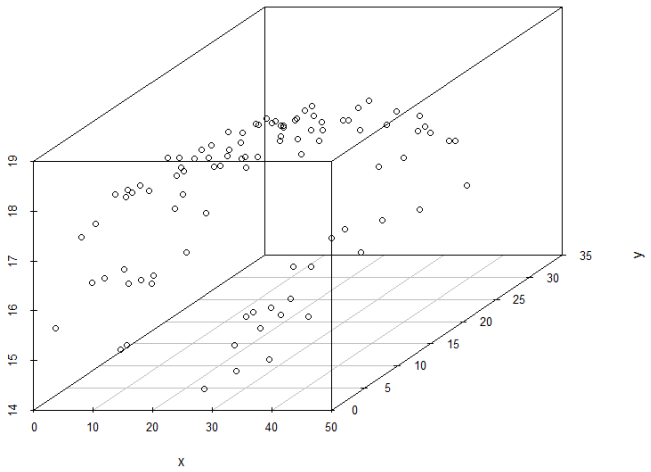

Figura 2: Resultado - Disposição dos dados

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

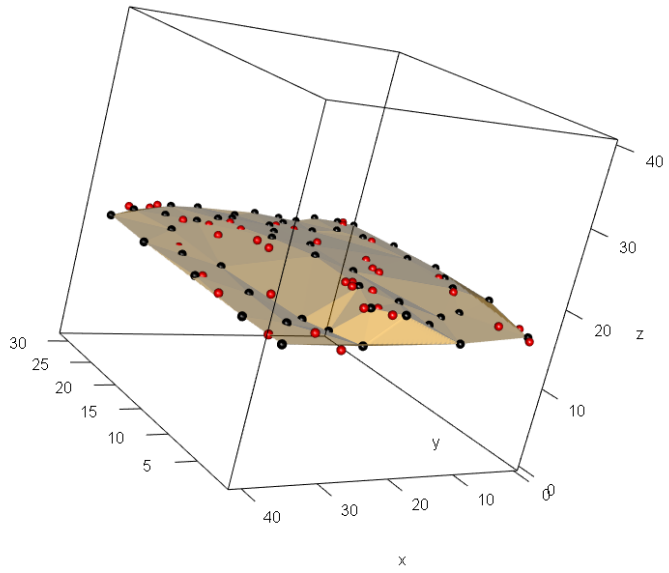

Figura 3: Resultado - Disposição dos dados - SURFACE

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)

```
d_tr=dados[1,]

trans_p = function(p, pontos){
  #matriz 14400 x 50 colunas(pontos gerados)
  result = matrix(0, nrow = nrow(dados), ncol = nrow(pontos))
  d_tr = dados[1,]
  result[1,] = reg_lin(d_tr, dados[1,], pontos)

  for(i in 2: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]
      }
    }
    result[i,] = reg_lin(d_tr, dados[i,], pontos)
  }
  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()
  for(i in 1:nrow(result)){
    a = (dados[i,] - result[i,])^2
    rmse_epoca[i] = sqrt(mean(a))
  }
  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)

```r
# 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

Tabela de Resultados (Erros Médios)

1	7939	7940	7914	7941	7913	7916	7938	7912	7927	8240
2	8241	8230	8231	8228	8232	8229	8243	8202	8203	8325
3	9281	9282	8243	9280	8214	8242	8241	8240	8215	8244
4	9282	9281	9280	8325	8243	8216	8214	8199	8240	8215
5	9281	9282	9280	8243	8241	8242	8202	8213	8200	8216
6	9281	9282	9280	8243	8214	8242	8241	8213	8239	8325
7	9282	9281	9280	8215	8243	8214	8202	8201	8213	8216
8	9282	9281	9280	8243	8202	8200	8216	8325	8201	8209
9	9282	9281	9280	8243	8242	8214	8241	9283	8202	8216

Figura 4: As épocas que apresentaram os 10 menores erros para cada probabilidade (linhas)

1	5289	5290	5291	5292	5276	5277	5293	5256	5294	5472
2	5289	5290	5291	5276	5292	5293	5277	5256	5294	5472
3	5276	5277	5289	5290	5291	5292	5256	5293	5294	5472
4	5289	5290	5291	5276	5277	5256	5292	5293	5294	5472
5	5276	5277	5289	5290	5291	5292	5256	5293	5294	5472
6	5276	5277	5289	5290	5291	5292	5293	5256	5294	5472
7	5276	5277	5289	5290	5291	5292	5293	5256	5294	5472
8	5276	5277	5289	5290	5291	5292	5293	5256	5294	5472
9	5276	5277	5289	5290	5291	5256	5292	5293	5294	5472

Figura 5: As épocas que apresentaram os 10 maiores erros para cada probabilidade (linhas)

Agora, mostramos os erros médios (Máximo, Médio e Mínimo) e em seguida a MED dos erros médios com a variância.

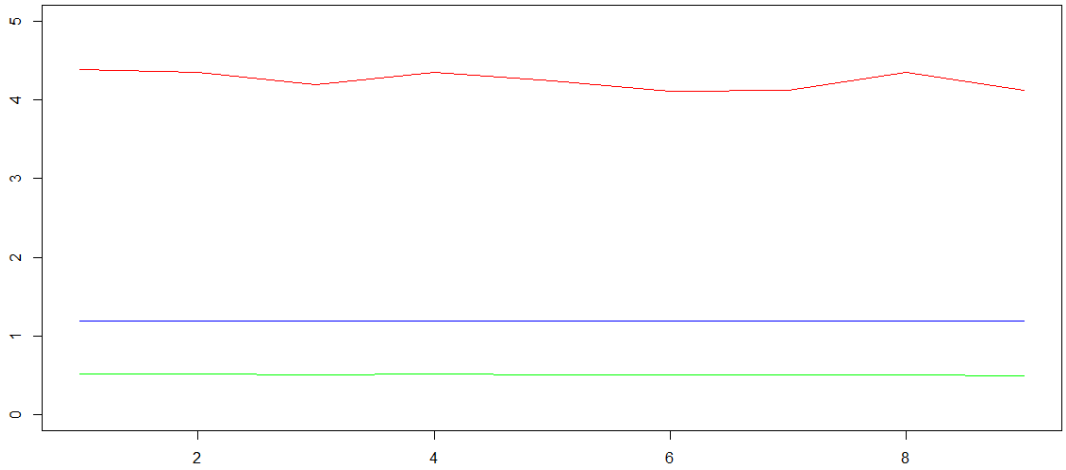

Figura 6: Erros médios (Máximo, Médio e Mínimo)

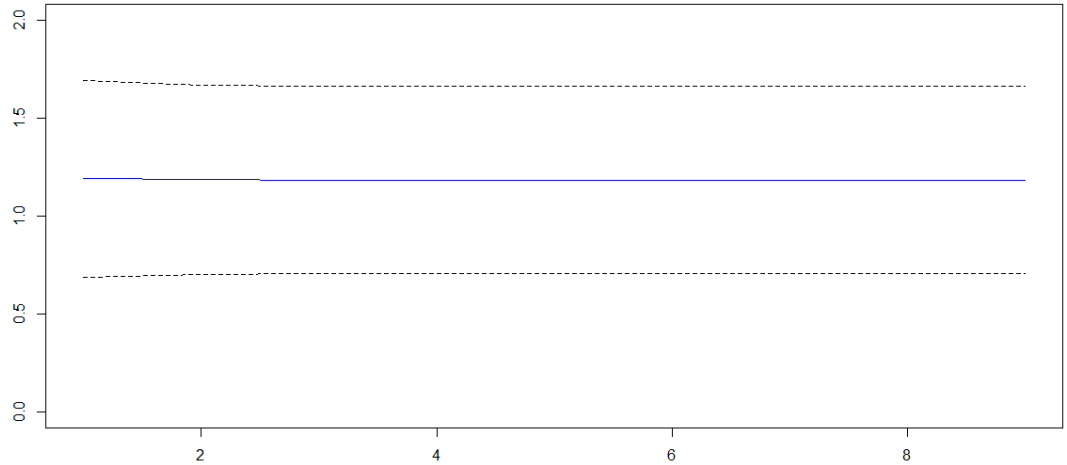

Figura 7: MED dos erros Médios + Variância

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)

```r
# 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

```r
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)

```r
# 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:

```r
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:

   Tabela de Resultados (Erros Médios)

1	1	2	6259	6260	6261	6234	6235	6233	6240	6251
2	1	6230	6241	6258	6239	6243	6240	27	6242	29
3	1	6160	6248	6159	6271	6249	6264	6245	6231	6263
4	1	6153	6236	6274	6265	6239	6254	6260	6257	6255
5	1	6257	6156	6237	21	52	6161	6146	6245	6260
6	1	53	54	6142	2	6268	6249	21	6260	6251
7	1	53	14	19	20	54	6169	6272	6146	6166
8	1	6169	7	52	6275	6231	6145	3	6162	22
9	1	53	2	8	14	22	6171	15	5	4

Figura 8: As épocas que apresentaram os 10 menores erros para cada probabilidade (linhas)

1	5293	5294	5292	5291	5290	5289	5471	5473	5474	4259
2	5289	5297	5298	5299	5296	5300	5472	4259	9292	9294
3	4259	5472	5293	5295	5267	5303	5466	5258	5294	5488
4	5297	5296	5473	5293	5268	5269	5303	5466	5257	5295
5	5473	5268	5271	5270	5269	5466	5259	5258	3726	3746
6	5291	5290	5289	4261	4260	5475	5474	5476	5473	5470
7	5289	4259	5473	5466	3749	3717	5471	3718	3990	3991
8	5473	3725	4011	3990	3991	5471	3744	3743	3745	3734
9	3745	3744	3743	3733	3734	3742	3740	3746	3741	3737

Figura 9: As épocas que apresentaram os 10 maiores erros para cada probabilidade (linhas)

Agora, mostramos os erros médios (Máximo, Médio e Mínimo) e em seguida a MED dos erros médios com a variância.

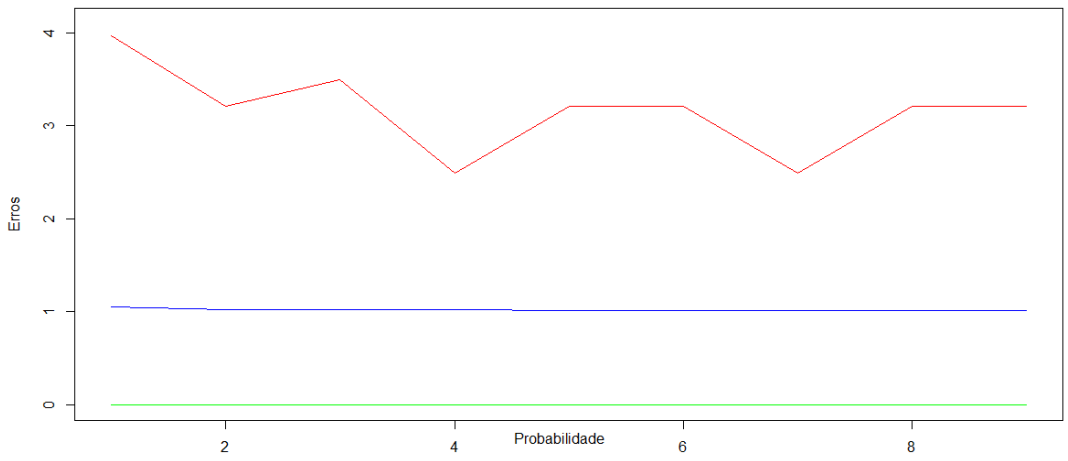

Figura 10: Erros médios (Máximo, Médio e Mínimo)

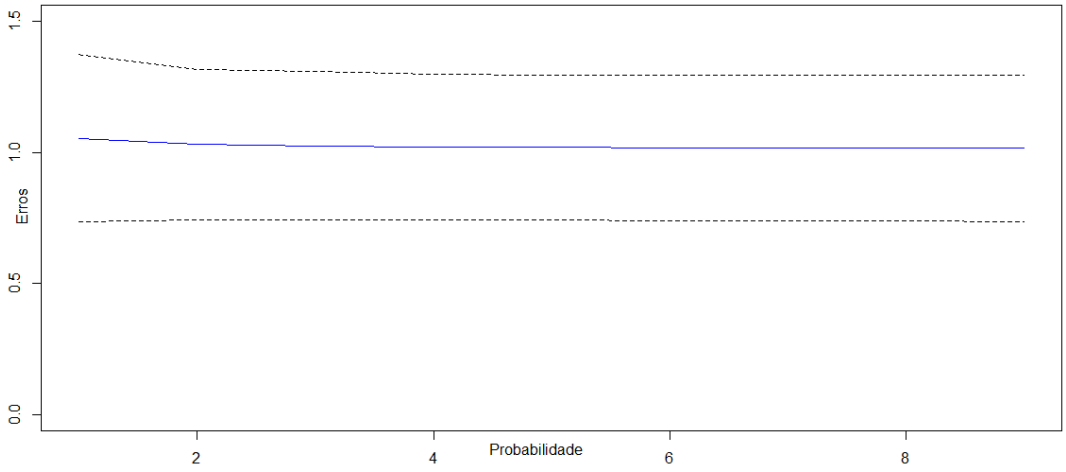

Figura 11: MED dos erros Médios + Variância

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
```

```
27
28    return(a)
29  }
```

2. Função de teste da RBF

```
1  teste_rbf = function(pontos, k, n_neuronios, w){
2    H = matrix(1, nrow=nrow(pontos), ncol = n_neuronios+1) # 50X11 - 1 Bias
3    means    = k$centers
4    clusters = k$cluster
5    v = array(0, c(2,2, n_neuronios))
6    for(i in 1:n_neuronios){
7      group = which(clusters %in% i) # Unir os do mesmo grupo
8      v[,,i] = var(locs[group,])     # variancia do grupo
9    }
10
11   for(i in 1:nrow(pontos)){
12     for(j in 1:n_neuronios){
13       sigma = diag(1,2)
14       diag(sigma) = diag(v[,,j])
15       H[i,j+1] = exp(-(t(pontos[i,]-means[j,]) %*% solve(sigma) %*% (pontos[i
       ,]-means[j,])))
16     }
17   }
18   return(t(H%*%w))
19 }
```

3. Função de treinamento da RBF

```
1  treino_rbf = function(y, n_neuronios, k){
2    H = matrix(1, nrow=nrow(locs), ncol = n_neuronios+1) # Matriz de 1 - 52x11
       (1 - Bias)
3    means    = k$centers
4    clusters = k$cluster
5
6    # Variancia dos grupos
7    v = array(0, c(2,2, n_neuronios))
8    for(i in 1:n_neuronios){
9      group = which(clusters %in% i)
10     v[,,i] = var(locs[group,])
11   }
12
13   # H
14   for(i in 1:nrow(locs)){
15     for(j in 1:n_neuronios){
16       sigma = diag(1,2)
17       diag(sigma) = diag(v[,,j])
18       H[i,j+1] = exp(-(t(locs[i,]-means[j,]) %*% solve(sigma) %*% (locs[i,]-
       means[j,])))
19     }
20   }
21   # W
22   return(MASS::ginv(t(H)%*%H)%*%t(H)%*%t(y))
23 }
```

4. Execução:

```
1  sigma  = estima_sigma(dados)
2  pontos = gera_pontos(50)
3
4  # Funcao Kernel (Formula 2)
```

```r
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 (com K = 10):

   Tabela de Resultados (Erros Médios)

1	11302	11301	11351	11298	11290	11294	11297	11419	11354	11403
2	11555	11557	11652	11575	11529	11640	11642	11537	11540	11093
3	11839	11836	11793	11838	11812	11811	11789	11791	11787	11810
4	10942	11199	11198	10936	11200	11195	10937	11201	10946	11193
5	10868	10869	10799	10827	10871	10800	10818	10816	10928	10917
6	10943	10942	10922	10799	10801	10807	10804	11178	10951	10950
7	10933	10931	10932	10868	10876	11031	10937	10939	10807	10942
8	10829	10828	10822	10851	10807	10827	10922	10847	10796	10792
9	11260	11159	11158	11270	11272	11269	11274	11273	11290	11268

Figura 12: As épocas que apresentaram os 10 menores erros para cada probabilidade (linhas)

1	5289	7206	7207	7202	7203	7208	7204	7212	7201	7209
2	5290	5289	5291	5293	5292	5294	7142	7145	7144	7141
3	5473	5257	7202	7205	7204	7201	7206	7207	7203	7178
4	5293	7202	7204	7203	7205	7173	7174	7201	7206	7178
5	5289	5290	5472	7204	7202	7203	7205	7201	7206	7172
6	5290	5289	7202	7204	7203	7201	7205	7206	7207	7200
7	7202	7172	7203	7173	7204	7201	7174	7205	7171	7206
8	7204	7172	7173	7202	7174	7203	7205	7171	7178	7146
9	4259	7202	7204	7203	7205	7201	7206	7173	7172	7174

Figura 13: As épocas que apresentaram os 10 maiores erros para cada probabilidade (linhas)

Agora, mostramos os erros médios (Máximo, Médio e Mínimo) e em seguida a MED dos erros médios com a variância.

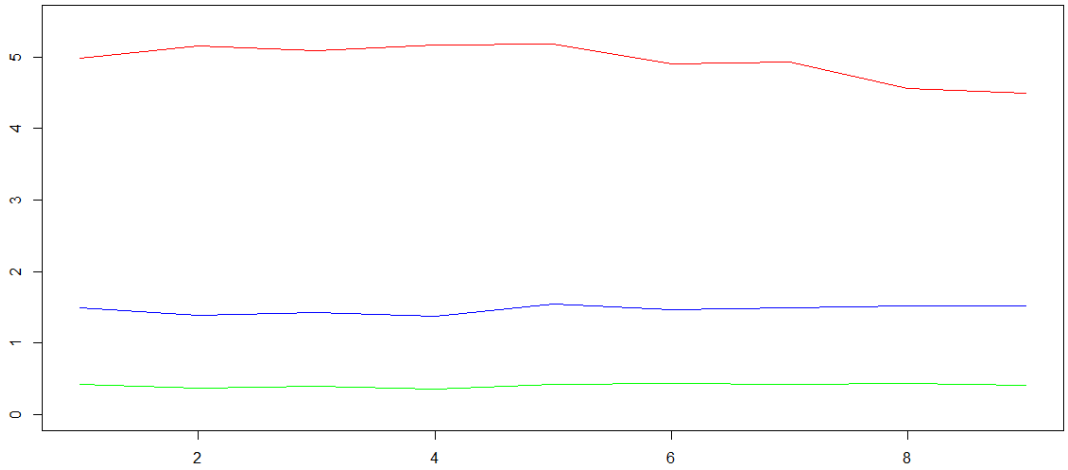

Figura 14: Erros médios (Máximo, Médio e Mínimo)

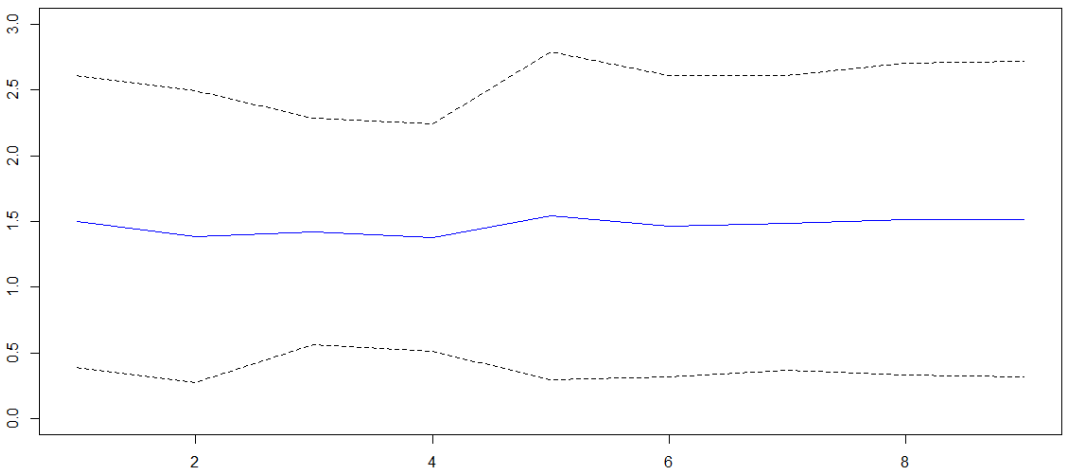

Figura 15: MED dos erros Médios + Variância

Observamos valores que fugiam dos padrões dos dados, e ao analisarmos encontramos alguns outliers (como podemos ver na Figura 16), Nas Figuras 17 e 18, tratamos esses outliers pegando um valor próximo ao da vizinhança, apenas para teste e obtivemos os seguintes resultados, onde tivemos uma pequena diminuição nos valores dos erros (quase insignificante para o erro médio, que é nosso foco).

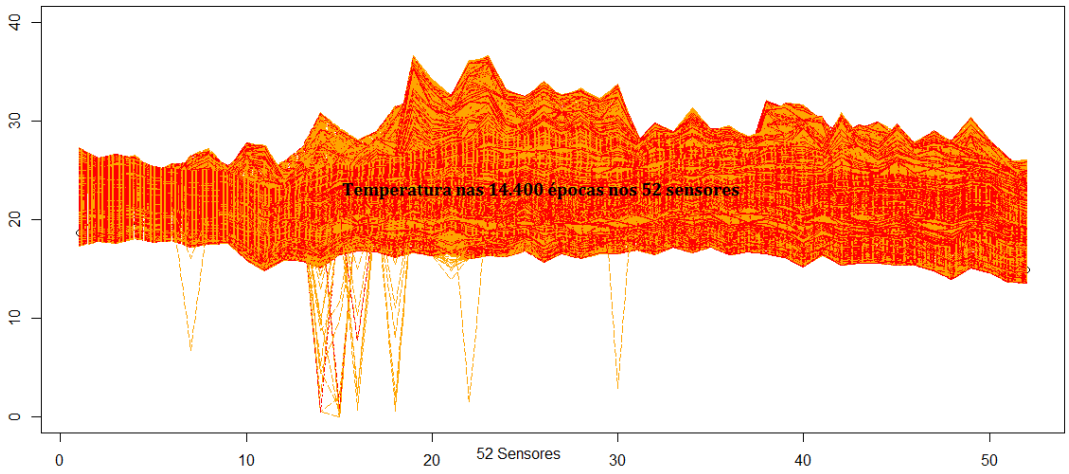

Figura 16: Outliers - Visualizando

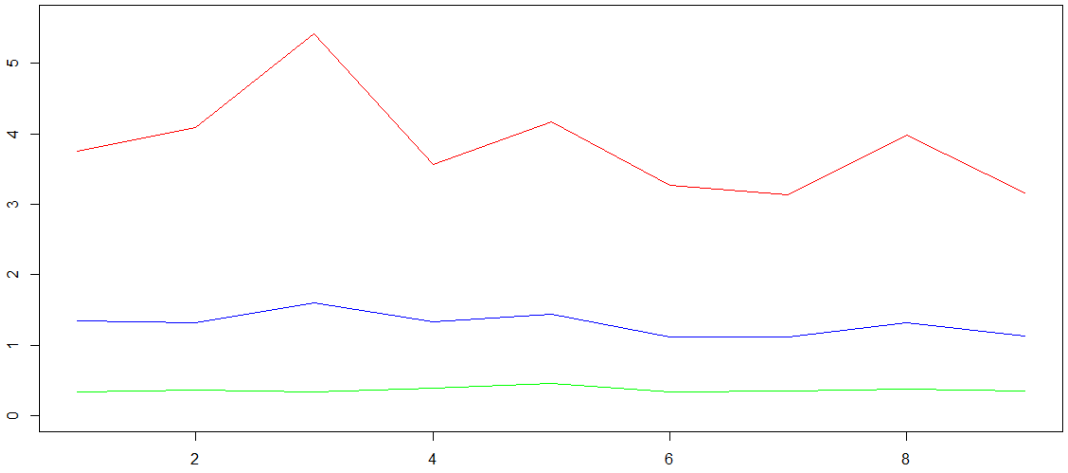

Figura 17: Erros médios (Máximo, Médio e Mínimo) - Sem Outliers

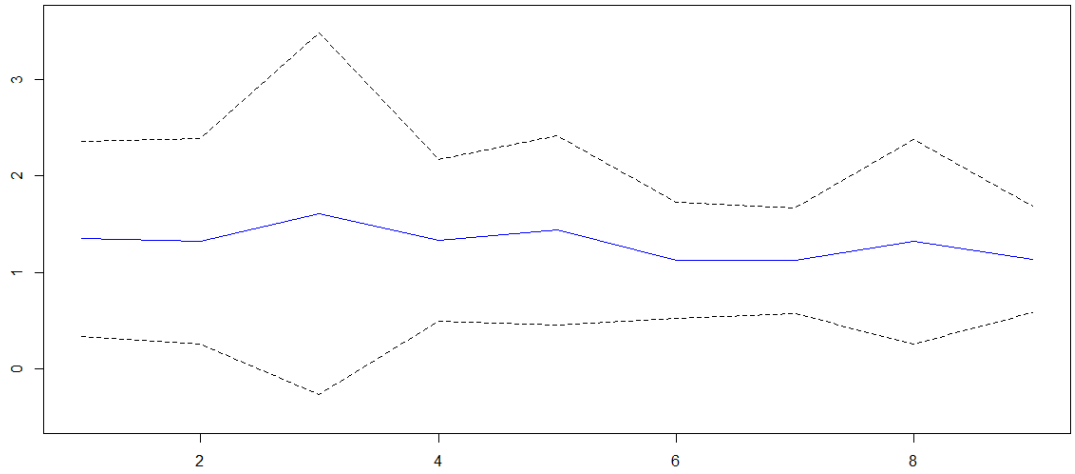

Figura 18: MED dos erros Médios + Variância - Sem Outliers

3.5 Regressão - Processo Gaussiano (item e)

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

Para o problema, iremos utilizar o pacote do R denominado: GPFit (https://cran.r-project.org/web/packages/GPfit/index.html), com método exponencial com valor de potência = 1.95, e os demais parâmetros são os padões sugeridos pela biblioteca.

Observacao: O pacote é muito lento para executar no R, esta levando cerca de 24hs para gerar uma probablidade, os resultados desmontrados abaixo esta da probabilidade 0.1 a 0.6. (as demais estamos providênciando)

Algumas configurações do pacote e formulas:

$$y(x) = \mu + Z(x), xin[0,1]^d \tag{5}$$

Onde Z(x) é um processo gaussiano com média 0

$$Var(Z(x_i)) = \sigma^2 \tag{6}$$

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:

```
# Executando o Algoritmo
sigma  = estima_sigma(dados) # Funcao ja descrita anteriormente
pontos = gera_pontos(50)     # Funcao ja descrita anteriormente

# Funcao Kernel (Formula 2)
kernel_gauss_gt = function(locs , x, sigma){
  m_x = t(replicate(52, x))
  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)
}

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

# Importando a biblioteca GPFIT
library('GPfit')
library(lhs)

# Funcao para calcular a probabilidade (probabilidade de quem vai ou nao
    transmitir)
# Quanto maior a probabilidade, maior o numero de sensores (o "melhor")
# Processo Gaussiano
trans_p = function(p, pontos){
  d_tr = dados[1,]
  result = matrix(0, nrow = nrow(dados), ncol = nrow(pontos))

  ytreino = dados * 0
  ytreino[1,] = d_tr
  for(i in 2:nrow(dados)){
    for (j in 1:nrow(locs)) {
      if(runif(1)<p){
        d_tr[j] = dados[i,j]
      }
    }
    ytreino[i,] = d_tr

    GPmodel = GP_fit(locs,ytreino[i,]);

    GPprediction = predict.GP(GPmodel,pontos);

    result[i,] = GPprediction$Y_hat
    print(i)
  }
  return(result)
}

# Monta resultados 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)
```

Resultados:

Tabela de Resultados (Erros Médios)

As épocas que apresentaram os 10 menores erros para cada probabilidade(linhas)

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	5122	5117	5125	5116	5124	5123	5135	5121	5112	5127
2	5114	5123	5029	5122	5099	5119	5115	5107	5106	5100
3	5111	5112	5110	5119	5107	5118	5023	5109	10615	5106
4	5116	5115	5117	5054	5118	5119	5055	5111	5112	5056
5	5117	5116	5113	5112	5115	5114	5109	5111	5103	5102

As épocas que apresentaram os 10 maiores erros para cada probabilidade(linhas)

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	1	5466	5257	4259	5267	5293	6412	5467	6411	6410
2	1	5466	5257	5267	4259	5293	5467	5276	5277	5266
3	5466	5267	5257	4259	5292	5295	5276	5277	5294	5265
4	1	5466	5257	5293	5267	4259	5467	5292	5277	5276
5	1	5466	5267	5257	4260	5467	5277	5489	5276	5266

Figura 19: As épocas que apresentaram os 10 menores e menores erros para cada probabilidade (linhas)

Agora, mostramos a MED dos erros médios.

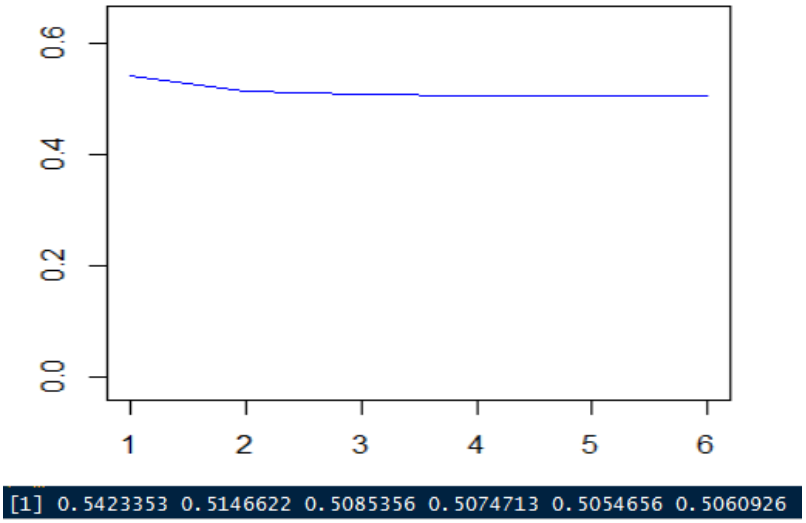

Figura 20: Erros médios - Média

4 Conclusões

A aplicação dos diversos modelos apresentaram pouca variação nos resultados, os melhores resultados foram obtidos com Processos Gaussianos. Ainda em relação a Processos Gaussianos, a biblioteca que usamos GPFIT, é lenta (cerca de 24hs para cada probabilidade) para gerar as probabilidades e fizemos até a probabilidade 0.6, onde daremos continuidade na geração para o artigo a ser publicado.

Observamos também alguns outliers, mas aplicando testes simples, vimos que não gera muito impacto.