

Lista 3

Lincoln Sousa

25/07/2020

Carregue o banco de dados saude.csv. Lembre colocar a variável Sexo como um factor. Ele contém diversas informações sobre a saúde de diversos pacientes:

- Sexo;
- Idade em anos;
- ALT: Altura em polegadas;
- Peso: Peso em libras;
- CINT: Circunferência da cintura em cm;
- TXPUL: Taxa de pulsação em batimentos por minuto;
- SIST: Pressão sanguínea sistólica em mmHg;
- DIAST: Pressão sanguínea diastólica em mmHg;
- DIF: Diferença da pressão sanguínea em mmHg;
- COL: colesterol in mg;
- IMC: índice de massa corporal;
- Perna: comprimento da parte superior da perna em cm;
- COTOV: largura do cotovelo em cm;
- Pulso: largura do pulso em cm;
- Braço: circunferência do braço em cm.

Crédito: U.S. Department of Health and Human Services, National Center for Health Statistics, Third National Health and Nutrition Examination Survey.

```
library(caret);library(dplyr);library(readr)
data = readr::read_delim("saude.CSV", ";",
  col_types = cols(Sexo = col_factor(levels = c())))
```

1. Realize o pré processamento (fora do train()) para identificar e remover:

(a) variáveis de variância zero ou quase zero;

```
caret::nearZeroVar(data,saveMetrics = T)
```

```
##      freqRatio percentUnique zeroVar  nzv
## Sexo    1.000000         2.50  FALSE FALSE
```

```
## Idade 1.250000      46.25 FALSE FALSE
## ALT 1.000000      77.50 FALSE FALSE
## Peso 1.000000     96.25 FALSE FALSE
## CINT 1.000000     83.75 FALSE FALSE
## TXPUL 1.272727     15.00 FALSE FALSE
## SIST 1.000000     48.75 FALSE FALSE
## DIAST 1.200000     43.75 FALSE FALSE
## DIF 1.200000     43.75 FALSE FALSE
## COL 1.000000     95.00 FALSE FALSE
## IMC 1.333333     81.25 FALSE FALSE
## Perna 1.000000     76.25 FALSE FALSE
## COTOV 1.166667     37.50 FALSE FALSE
## Pulso 1.000000     26.25 FALSE FALSE
## Braco 1.333333     76.25 FALSE FALSE
```

#Não possui nenhuma variável para ser removida

(b) variáveis com correlação absoluta acima de 0.75;

Como seria muito massante observar os gráficos de dispersão para as 14 variáveis, iremos realiar a correlação pelo método de pearson e também de spearman.

#Pearson

```
corsP = cor(data[,2:15],method = "pearson")
summary(corsP[upper.tri(corsP)])
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2489  0.1059  0.3068  0.3290  0.5307  0.9083
```

#Existem variáveis muito correlacionadas

```
caret::findCorrelation(corsP,cutoff = 0.75,verbose = T)
```

```
## Compare row 3  and column 4 with corr 0.908
## Means: 0.542 vs 0.345 so flagging column 3
## Compare row 4  and column 14 with corr 0.853
## Means: 0.472 vs 0.317 so flagging column 4
## Compare row 12  and column 13 with corr 0.802
## Means: 0.459 vs 0.29 so flagging column 12
## Compare row 14  and column 10 with corr 0.892
## Means: 0.361 vs 0.26 so flagging column 14
## All correlations <= 0.75
```

```
## [1] 3 4 12 14
```

```
corAltaP = caret::findCorrelation(corsP, cutoff = 0.75, names = T)
corAltaP
```

```
## [1] "Peso" "CINT" "COTOV" "Braco"
```

#Spearman

```
corsS = cor(data[,2:15],method = "spearman")
summary(corsS[upper.tri(corsS)])
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2301  0.1450  0.3536  0.3397  0.5367  0.9065
```

#Existem variáveis muito correlacionadas

```
caret::findCorrelation(corsS,cutoff = 0.75,verbose = T)
```

```
## Compare row 3 and column 4 with corr 0.9
## Means: 0.548 vs 0.352 so flagging column 3
## Compare row 4 and column 14 with corr 0.867
## Means: 0.473 vs 0.323 so flagging column 4
## Compare row 12 and column 13 with corr 0.79
## Means: 0.462 vs 0.297 so flagging column 12
## Compare row 14 and column 10 with corr 0.906
## Means: 0.377 vs 0.271 so flagging column 14
## All correlations <= 0.75

## [1] 3 4 12 14

corAltaS = caret::findCorrelation(corsS, cutoff = 0.75, names = T)
corAltaS
```

```
## [1] "Peso" "CINT" "COTOV" "Braco"
```

Por ambos os métodos as variáveis para serem removidas foram as mesmas.

```
#Serão removidas as variáveis Peso, CINT, COTOV e Braco
data = dplyr::select(data, - dplyr::all_of(corAltaP))
```

(c) variáveis com dependência linear.

```
combLin = caret::findLinearCombos(data[2:11])
combLin
```

```
## $linearCombos
## $linearCombos[[1]]
## [1] 6 4 5
##
##
## $remove
## [1] 6
```

```
#Foi encontrado que as colunas 4 5 e 6 são combinações lineares. da base de dados
#sem contar a variável sexo iríamos remover a variavel 6, como temos que adicionar
#a coluna sexo, iremos remover a variavel 7.
data = data[,-(combLin$remove+1)]
```

2. O objetivo é prever o Sexo em função das demais variáveis. Para realizar tal tarefa, precisamos decidir qual o melhor método de treinamento do classificador. Queremos utilizar Support Vector Machine (SVM), mas não sabemos qual o melhor kernel, linear, polinomial ou radial. Realize validação cruzada para verificar qual é o melhor entre os 3 métodos acima (svmLinear, svmPoly, svmRadial) para construir o classificador.

- Utilize o banco de dados pré-processado.
- Utilize a mesma metodologia que utilizamos para avaliar classificadores.
- Utilize o método k-fold repetido, com k=10 e 3 repetições.
- Lembre de fixar o set.seed(1903) antes de cada treinamento.

- Ao final, redija um texto justificando qual método elegeu como melhor para esse problema.

```
treino_metodo = caret::trainControl(method = "repeatedcv",
                                   number = 10,
                                   repeats = 3)

set.seed(1903)
svmLinear = caret::train(Sexo ~ .,
                         data = data,
                         method = "svmLinear",
                         trControl = treino_metodo)

set.seed(1903)
svmPoly = caret::train(Sexo ~ .,
                      data = data,
                      method = "svmPoly",
                      trControl = treino_metodo)

set.seed(1903)
svmRadial = caret::train(Sexo ~ .,
                        data = data,
                        method = "svmRadial",
                        trControl = treino_metodo)
```

Resultados descritivos

```
resultados = caret::resamples(list(Linear = svmLinear,
                                   Poly = svmPoly,
                                   Radial = svmRadial))

summary(resultados)
```

```
##
## Call:
## summary.resamples(object = resultados)
##
## Models: Linear, Poly, Radial
## Number of resamples: 30
##
## Accuracy
##      Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
## Linear 0.750  0.875  0.875 0.9083333      1    1    0
## Poly   0.750  0.875  1.000 0.9250000      1    1    0
## Radial 0.625  0.875  0.875 0.9125000      1    1    0
##
## Kappa
##      Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
## Linear 0.50   0.75   0.75 0.8166667      1    1    0
## Poly   0.50   0.75   1.00 0.8500000      1    1    0
## Radial 0.25   0.75   0.75 0.8250000      1    1    0
```

Obtiveram-se resultados bastante parecidos das métricas Kappa e Acurácia para os 3 modelos, há somente uma diferença considerável em relação a mediana, que favorece o modelo svmPoly.

Comparação de tempo de execução

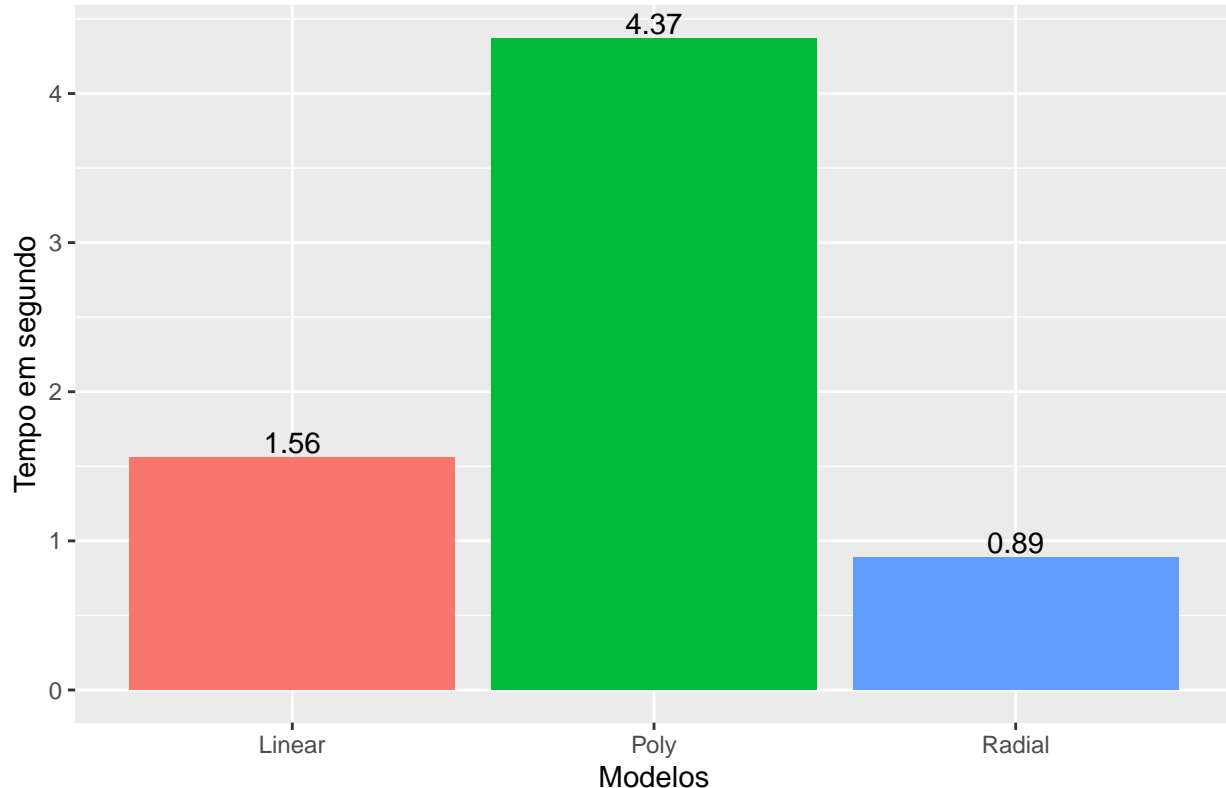
```
library(ggplot2)
ggplot(resultados$timings, aes(x = row.names(resultados$timings),
```

```

y = Everything,
fill = row.names(resultados$timings)) +
geom_bar(stat = "identity") +
labs(title = "Comparação de tempo de processamento entre os modelos de SVM",
y = "Tempo em segundo",
x = "Modelos") +
geom_text(aes(label = sprintf("%.2f", resultados$timings$Everything),
y= resultados$timings$Everything),vjust = -0.2)+
theme(legend.position = "none")

```

Comparação de tempo de processamento entre os modelos de SVM



Apesar do modelo svmPoly ter apresentado melhor mediana, ele é o que mais demora para ser executado em relação aos demais.

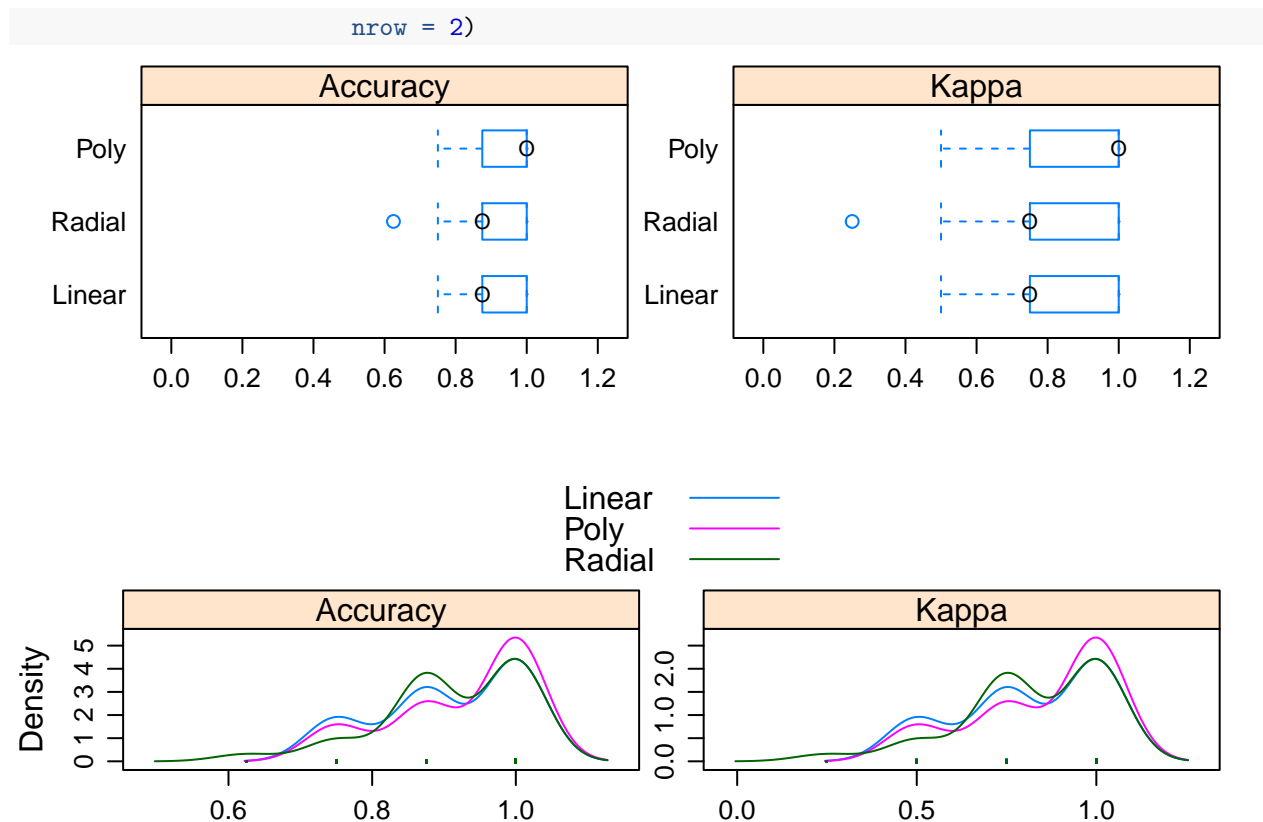
Comparação visual de variabilidade

```

escalas = list(x=list(relation="free" ),
               y=list(relation="free" ))

gridExtra::grid.arrange(
  lattice::bwplot(resultados,
                  scales = escalas,
                  pch = "0",
                  xlim = c(0,1.2)),
  lattice::densityplot(resultados,
                      scales = escalas,
                      pch = ".",
                      auto.key = TRUE),

```



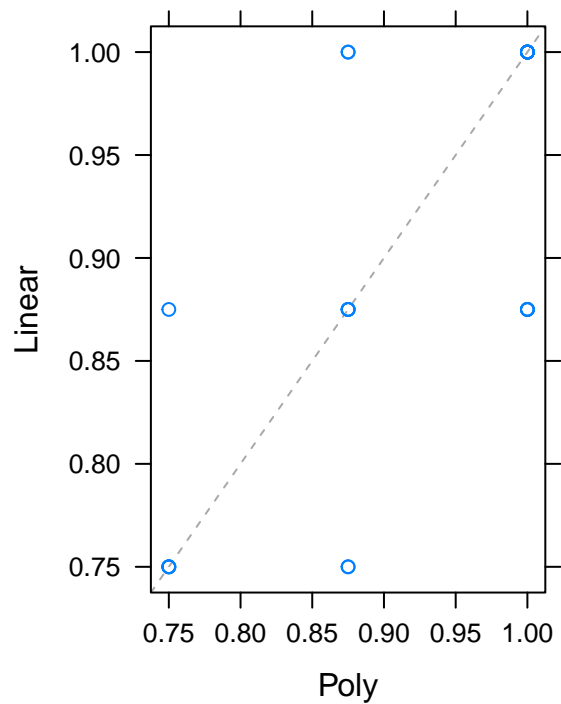
Apesar de possuir boxplots bastante parecidos, como já vimos, o modelo svmPoly possui as melhores medianas para as métricas testadas. Pelo gráfico de densidade, é possível notar que o modelo svmPoly também leva vantagem com comparação aos demais, tendo em vista que se concentra mais para valores maiores das métricas. Por essas observações, pode-se concluir que o modelo svmPoly é o modelo que melhor representa os dados empiricamente.

Comparação 2 a 2 dos kfold

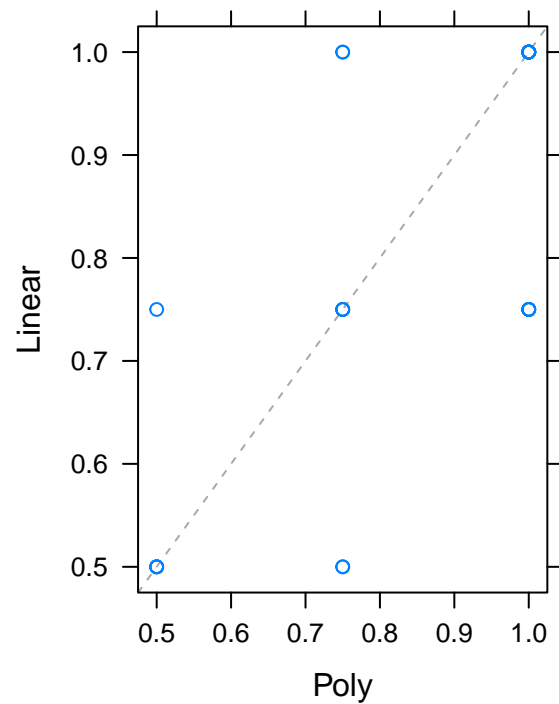
```
gridExtra::grid.arrange(
  lattice::xyplot(resultados, models = c("Linear","Poly"), metric = "Accuracy"),
  lattice::xyplot(resultados, models = c("Linear","Poly"), metric = "Kappa"),
  ncol = 2,
  top = "Comparando métricas para cada fold entre svmLinear E svmPoly"
)
```

Comparando métricas para cada fold entre svmLinear E svmPoly

Accuracy



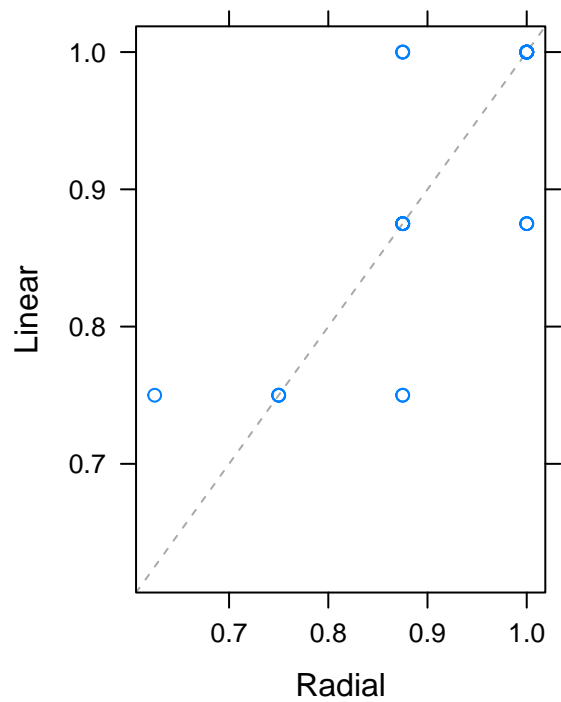
Kappa



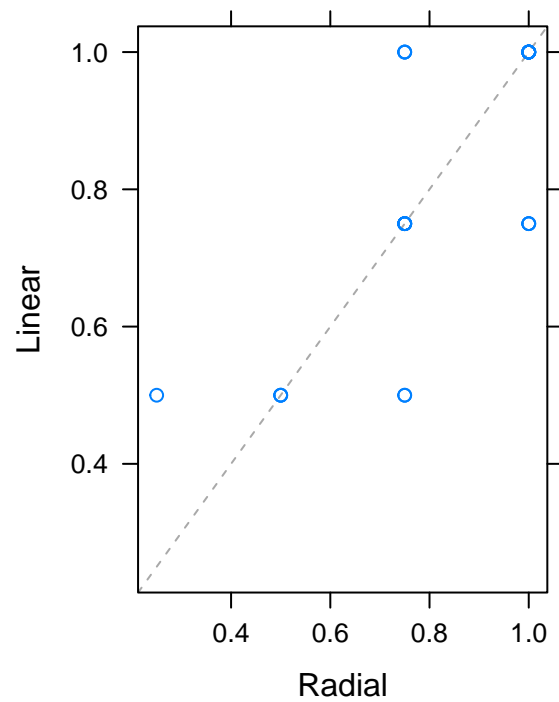
```
gridExtra::grid.arrange(  
  lattice::xyplot(resultados, models = c("Linear", "Radial"), metric = "Accuracy"),  
  lattice::xyplot(resultados, models = c("Linear", "Radial"), metric = "Kappa"),  
  ncol = 2,  
  top = "Comparando métricas para cada fold entre svmLinear E svmRadial"  
)
```

Comparando métricas para cada fold entre svmLinear E svmRadial

Accuracy

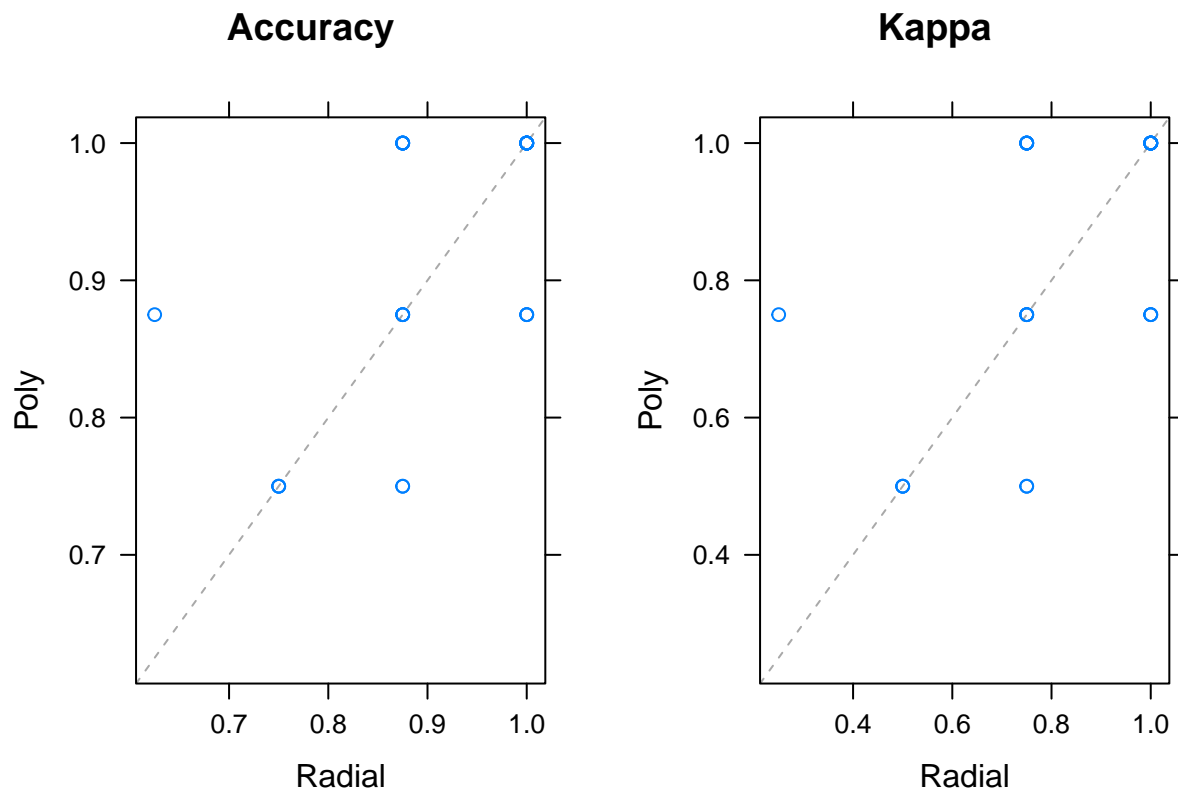


Kappa



```
gridExtra::grid.arrange(  
  lattice::xyplot(resultados, models = c("Poly","Radial"), metric = "Accuracy"),  
  lattice::xyplot(resultados, models = c("Poly","Radial"), metric = "Kappa"),  
  ncol = 2,  
  top = "Comparando métricas para cada fold entre svmPoly E svmRadial"  
)
```


Comparando métricas para cada fold entre svmPoly E svmRadial



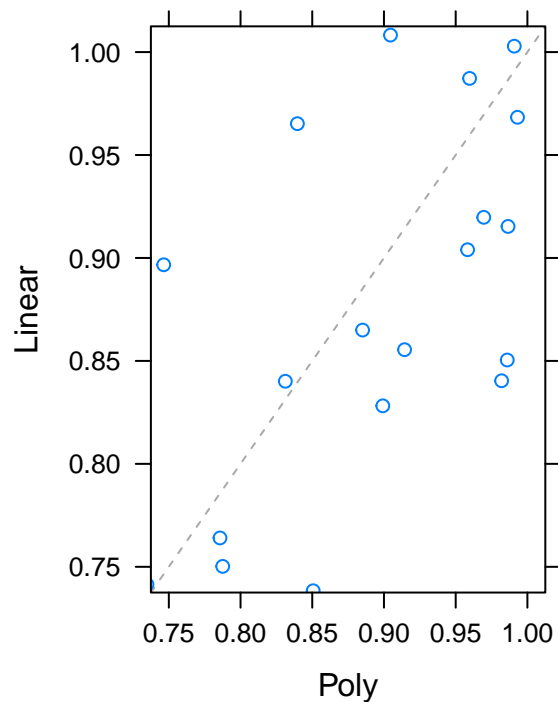
Vários folds obtiveram valores parecidos na comparação dois a dois para as estatísticas Acurácia e Kappa de todos os modelos. Por esse motivo, a interpretação desse tipo de gráfico fica imprecisa, pois vários pontos se sobrepõem e não é possível identificar visualmente qual dos modelos comparados obtiveram as melhores estatísticas por folds.

Comparação 2 a 2 dos kfold com jitter

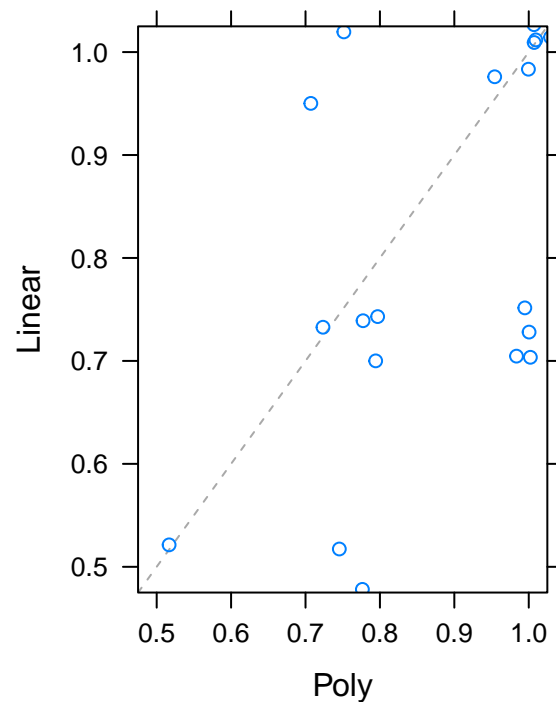
```
gridExtra::grid.arrange(
  lattice::xyplot(resultados, models = c("Linear", "Poly"), metric = "Accuracy",
    jitter.x = T, jitter.y = T, amount = 0.05, factor = 0.05),
  lattice::xyplot(resultados, models = c("Linear", "Poly"), metric = "Kappa",
    jitter.x = T, jitter.y = T, amount = 0.05, factor = 0.05),
  ncol = 2,
  top = "Comparando métricas para cada fold entre svmLinear E svmPoly"
)
```

Comparando métricas para cada fold entre svmLinear E svmPoly

Accuracy



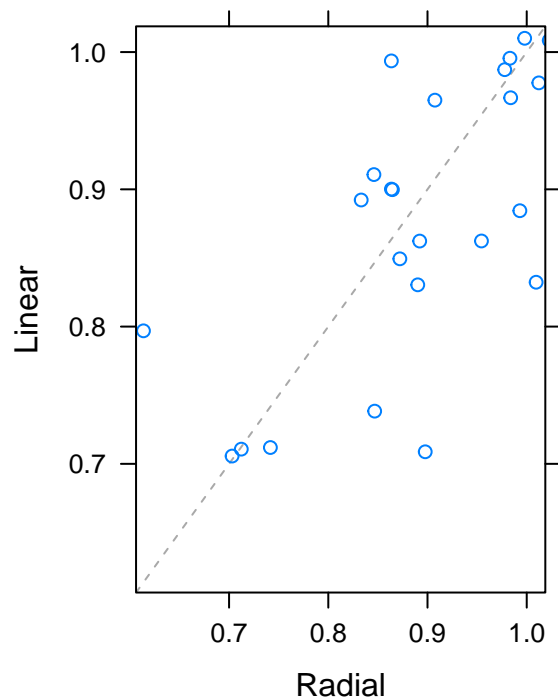
Kappa



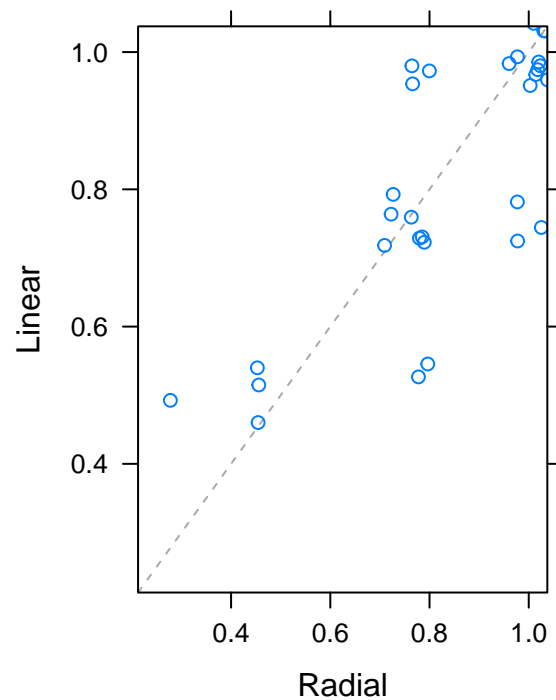
```
gridExtra::grid.arrange(
  lattice::xyplot(resultados, models = c("Linear", "Radial"), metric = "Accuracy",
    jitter.x = T, jitter.y = T, amount = 0.05, factor = 0.05),
  lattice::xyplot(resultados, models = c("Linear", "Radial"), metric = "Kappa",
    jitter.x = T, jitter.y = T, amount = 0.05, factor = 0.05),
  ncol = 2,
  top = "Comparando métricas para cada fold entre svmLinear E svmRadial"
)
```

Comparando métricas para cada fold entre svmLinear E svmRadial

Accuracy

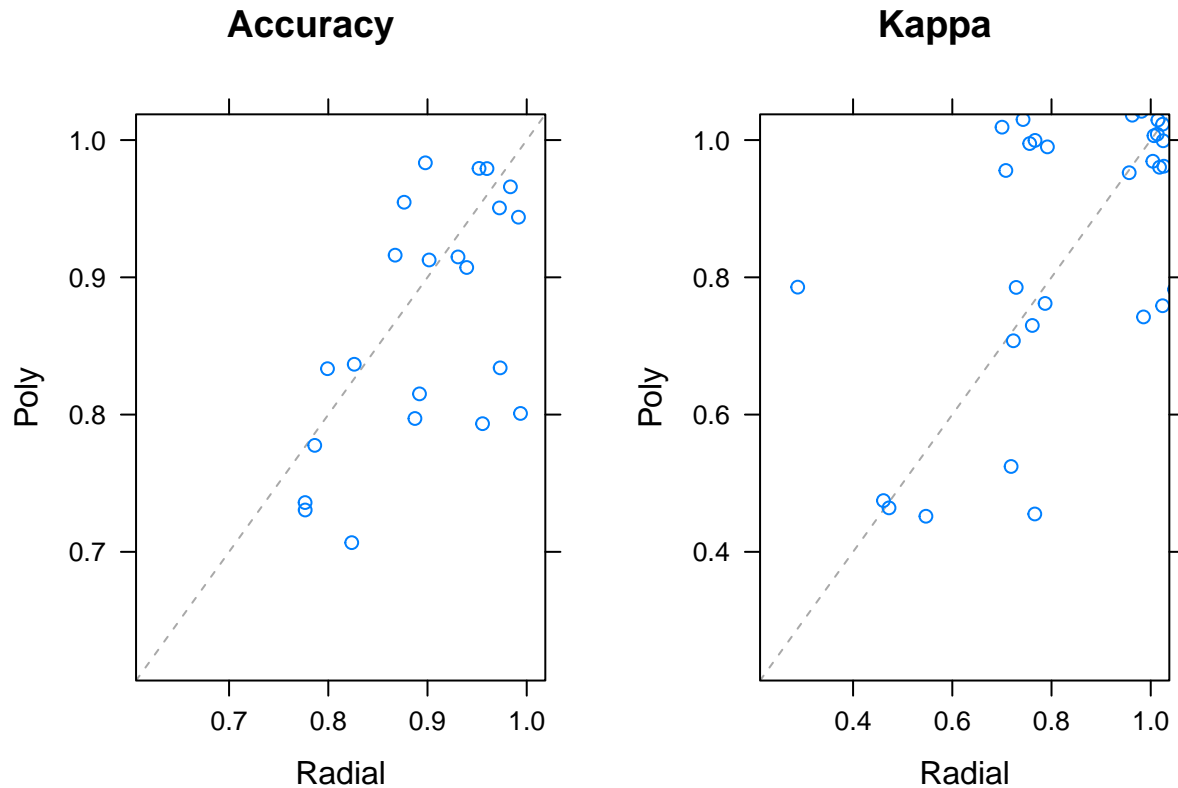


Kappa



```
gridExtra::grid.arrange(
  lattice::xyplot(resultados, models = c("Poly","Radial"), metric = "Accuracy", jitter.x = T,
    jitter.y = T, amount = 0.1, factor = 0.05),
  lattice::xyplot(resultados, models = c("Poly","Radial"), metric = "Kappa", jitter.x = T,
    jitter.y = T, amount = 0.05, factor = 0.05),
  ncol = 2,
  top = "Comparando métricas para cada fold entre svmPoly E svmRadial"
)
```

Comparando métricas para cada fold entre svmPoly E svmRadial



Foi utilizado o argumento jitter.x e jitter.y como verdadeiro para que o seja aplicado o efeito que faça os pontos “se espalharem” nos eixos x e y para que não fiquem sobrepostos, assim, é possível ter uma ideia no comportamento dos folds. O argumento amount e factor = 0.05 foi feito de forma que essa variação dos pontos sobrepostos se movimentem até 0.05 em x e y do seu ponto original.

É possível observar pelos gráficos que a comparação das métricas por folds entre os modelos 2 a 2 é bastante parecida. Aparentemente nenhum modelo se destaca.

Testes de hipótese para diferença das métricas dos modelos dois a dois

```
difs = diff(resultados)
summary(difs)

##
## Call:
## summary.diff.resamples(object = difs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##      Linear Poly      Radial
## Linear      -0.016667 -0.004167
## Poly    0.6339      0.012500
## Radial 1.0000 1.0000
##
## Kappa
##      Linear Poly      Radial
```

```
## Linear      -0.033333 -0.008333
## Poly    0.6339      0.025000
## Radial 1.0000 1.0000
```

Adotando um nível de significância $\alpha = 0.05$, obteve-se que $\alpha > 0.05$ para todos os testes realizados, isso é, não rejeita-se nenhuma das hipóteses nulas H_0 . Assim conclui-se que há evidências estatísticas que para a comparação dois a dois das métricas de todos os 3 modelos, eles possuem diferença = 0, ou seja, são iguais.

Mesmo obtendo pelos testes de hipótese que os modelos possuem métricas iguais, ira ser analisado os dados empíricos obtidos na validação cruzado com o método de treinamento executado. Mesmo com um tempo de execução maior, o modelo svmPoly obteve melhores resultados para as métricas acurácia e kappa, e por isso, será o modelo escolhido.

3. Uma vez escolhido o melhor método classificador, vamos criar o classificador de fato. Fixe a semente no valor 100. Crie amostras treino (75%) e teste (25%), utilizando os dados pré- processados. Utilize o método de re-amostragem bootstrap, com 20 re-amostragens. Treine o classificador pelo método escolhido e calcule uma estimativa para o erro fora da amostra.

```
set.seed(100)
treino_index = caret::createDataPartition(data$Sexo,p = 0.75, list = F)
metodo_treino_boot = caret::trainControl(method = "boot", number = 20)
treino = data[treino_index,]

## Warning: The `i` argument of `[`() can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

teste = data[-treino_index,]

set.seed(100)
modelo_boot_svmPoly = caret::train(Sexo ~ .,
                                   data = treino,
                                   method = "svmPoly",
                                   trControl = metodo_treino_boot)

# Fazendo a predição
pred_boot_svmPoly = predict(modelo_boot_svmPoly,
                             teste)

# Matriz de confusão para a amostra teste
resultado_teste = caret::confusionMatrix(pred_boot_svmPoly,
                                           teste$Sexo)

resultado_teste

## Confusion Matrix and Statistics
##
##              Reference
## Prediction M F
##              M 8 1
```

```
##           F 2 9
##
##           Accuracy : 0.85
##           95% CI : (0.6211, 0.9679)
##           No Information Rate : 0.5
##           P-Value [Acc > NIR] : 0.001288
##
##           Kappa : 0.7
##
##           McNemar's Test P-Value : 1.000000
##
##           Sensitivity : 0.8000
##           Specificity : 0.9000
##           Pos Pred Value : 0.8889
##           Neg Pred Value : 0.8182
##           Prevalence : 0.5000
##           Detection Rate : 0.4000
##           Detection Prevalence : 0.4500
##           Balanced Accuracy : 0.8500
##
##           'Positive' Class : M
##
```

4. Queremos entender o impacto do pré-processamento no classificador. Repita o item 3. com os dados brutos, ou seja, antes de realizar o pré-processamento. O que foi possível observar?

```
data_bruto = readr::read_delim("saude.CSV", ";",
  escape_double = FALSE, col_types = cols(Sexo = col_factor(levels = c())) ,
  trim_ws = TRUE)

set.seed(100)
treino_index = caret::createDataPartition(data_bruto$Sexo, p = 0.75, list = F)
metodo_treino_boot = caret::trainControl(method = "boot", number = 20)
treino = data_bruto[treino_index,]
teste = data_bruto[-treino_index,]

set.seed(100)
modelo_boot_svmPoly = caret::train(Sexo ~ .,
  data = treino,
  method = "svmPoly",
  trControl = metodo_treino_boot)

# Fazendo a predição
pred_boot_svmPoly = predict(modelo_boot_svmPoly,
  teste)

# Matriz de confusão para a amostra teste
resultado_teste = caret::confusionMatrix(pred_boot_svmPoly,
  teste$Sexo)

resultado_teste
```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  F
##           M 10  1
##           F  0  9
##
##           Accuracy : 0.95
##           95% CI : (0.7513, 0.9987)
##           No Information Rate : 0.5
##           P-Value [Acc > NIR] : 2.003e-05
##
##           Kappa : 0.9
##
## Mcnemar's Test P-Value : 1
##
##           Sensitivity : 1.0000
##           Specificity : 0.9000
##           Pos Pred Value : 0.9091
##           Neg Pred Value : 1.0000
##           Prevalence : 0.5000
##           Detection Rate : 0.5000
##           Detection Prevalence : 0.5500
##           Balanced Accuracy : 0.9500
##
##           'Positive' Class : M
##

```

Observa-se que o classificador criado utilizando os dados sem realizar as técnicas de pré-processamento obteve métricas maiores do que o criado utilizando, o que não era esperado. A etapa de pré-processamento foi realizada com o objetivo de melhorar a qualidade dos dados, mas mesmo assim, foi visto que o classificador criado sem utilização dessas técnicas obteve um desempenho melhor.