k Vizinhos Mais Próximos

EST0133 - Introdução à Modelagem de Big Data

Marcus Nunes
https://introbigdata.org/
https://marcusnunes.me/

Universidade Federal do Rio Grande do Norte

Introdução

Introdução

- Também conhecido como k nearest neighbor (kNN)
- Método não-paramétrico de classificação
- Depende de uma votação para definir a que classe cada observação pertence
- Não existe um modelo explícito para os resultados obtidos
- É um método de aprendizagem supervisionada

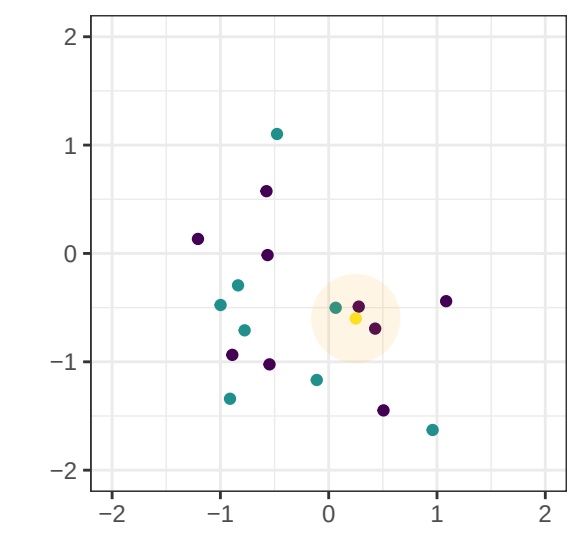
Algoritmo

Algoritmo

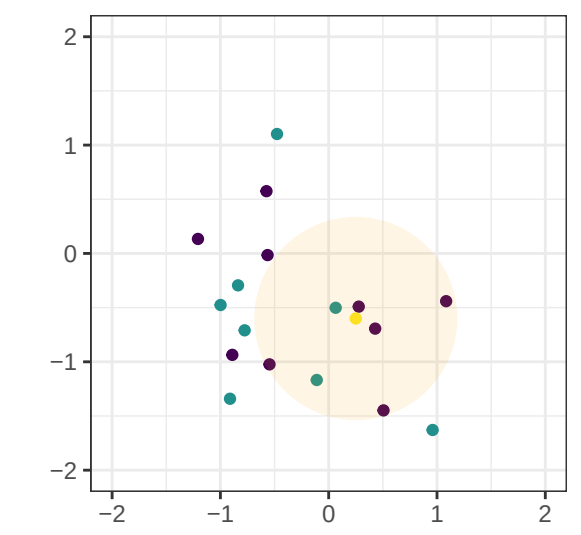
- Dado um inteiro positivo k, uma observação $\boldsymbol{x}$ e uma distância d, o algoritmo calcula a distância d entre $\boldsymbol{x}$ e cada ponto no conjunto de treinamento
- Assim é possível criar um conjunto A formado pelos k pontos mais próximos de $\boldsymbol{x}$
- A partir disso calculamos a probabilidade condicional de $\boldsymbol{x}$ pertencer à classe j

$$P(y = j|X = \boldsymbol{x}) = \frac{1}{k} \sum_{i \in A} I(y^{(i)} = j)$$

Algoritmo - 3 Vizinhos Mais Próximos



Algoritmo - 7 Vizinhos Mais Próximos



Algoritmo

1. Encontre os k vizinhos mais próximos do ponto de interesse
2. Para uma tarefa de classificação, calcule a maioria de classes dentro da vizinhança dos k pontos
3. Para uma tarefa de regressão, calcule a média dos valores dos k vizinhos

Métricas de Avaliação

		Referência	
		Positivo	Negativo
Predição	Positivo	a	b
	Negativo	c	d

- Acurácia: $p_0 = \frac{a+d}{a+b+c+d}$
- Sensitividade: $\frac{a}{a+c}$
- Especificidade: $\frac{d}{b+d}$
- Curva ROC: Sensitividade (Verdadeiros Positivos) vs. 1-Especificidade (Falsos Positivos)

Métricas de Avaliação

		Referência	
		Positivo	Negativo
Predição	Positivo	a	b
	Negativo	c	d

- Kappa:

$$p_{\text{Sim}} = \frac{a+b}{a+b+c+b} \times \frac{a+c}{a+b+c+b}$$

$$p_{\text{Não}} = \frac{c+d}{a+b+c+b} \times \frac{b+d}{a+b+c+b}$$

$$p_e = p_{\text{Sim}} + p_{\text{Não}}$$

$$\kappa = \frac{p_0 - p_e}{1 - p_e}$$

Exemplo

Exemplo

- Vamos utilizar o conjunto de dados `iris`
- Queremos criar um modelo que classifique as espécies de plantas baseado nas suas características físicas
- Vamos utilizar o algoritmo *k*NN para isto

Exemplo

```
> # pacotes
>
> library(tidymodels)
> theme_set(theme_bw())
> library(ggfortify)
> library(GGally)
```

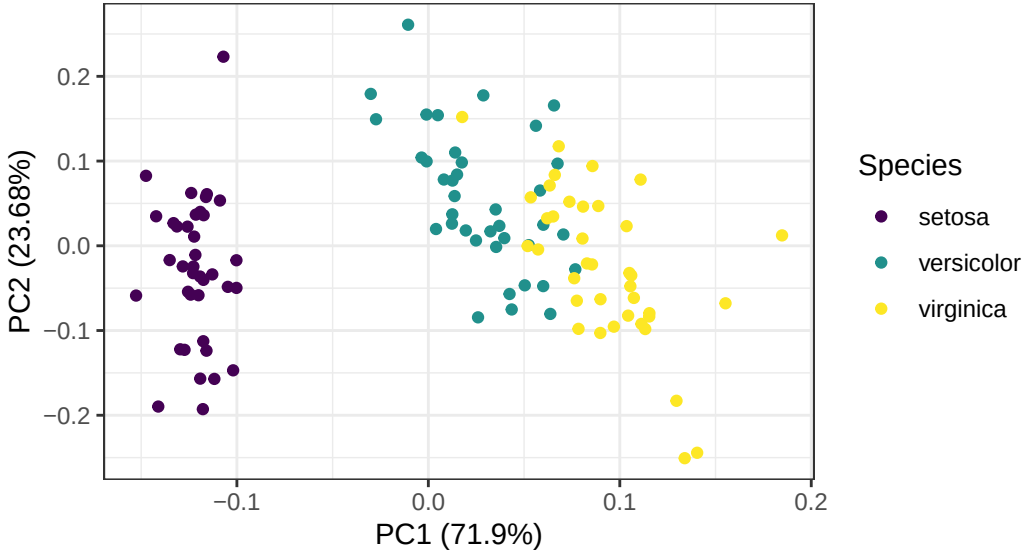
Exemplo

```
> # semente aleatoria
>
> set.seed(1234)
>
> # 75% dos dados como treino
>
> iris_split <- initial_split(iris, prop = .75, strata = Species)
>
> # criar os conjuntos de dados de treino e teste
>
> iris_treino <- training(iris_split)
>
> iris_teste  <- testing(iris_split)
```

Exemplo

```
> # eda
>
> autoplot(prcomp(iris_treino[, -5], center = TRUE, scale. = TRUE),
+          data = iris_treino,
+          colour = "Species") +
+   scale_colour_viridis_d()
```

Exemplo


PC2 (23.68%)
0.2
0.1
0.0
−0.1
−0.2
−0.1
0.0
0.1
0.2
PC1 (71.9%)
Species
setosa
versicolor
virginica

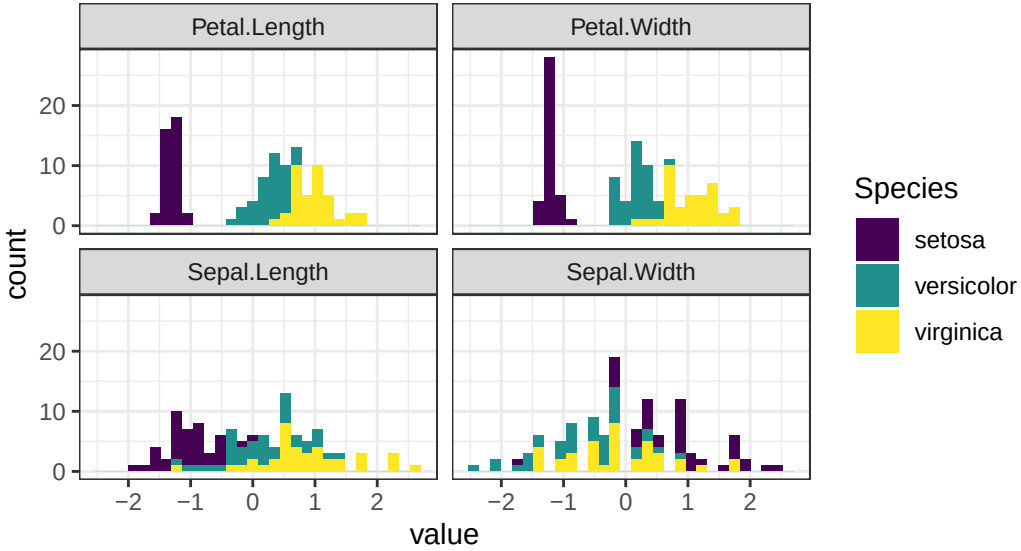
Exemplo

```
> # pre-processamento
>
> iris_rec <-
+   recipe(Species ~ .,
+          data = iris_treino) %>%
+   # remover observacoes de modo que todos os niveis de Species
+   # fiquem com o mesmo numero de observacoes
+   themis::step_downsample(Species) %>%
+   # center/scale
+   step_center(-Species) %>%
+   step_scale(-Species) %>%
+   # funcao para aplicar a transformacao aos dados
+   prep()
```

Exemplo

```
> # aplicar a transformacao aos dados
>
> iris_treino_t <- juice(iris_rec)
>
> # verificar o resultado do processamento de dados
>
> iris_treino_t %>%
+   pivot_longer(-Species) %>%
+   ggplot(., aes(fill = Species)) +
+   geom_histogram(aes(value)) +
+   facet_wrap(~ name) +
+   scale_fill_viridis_d()
```

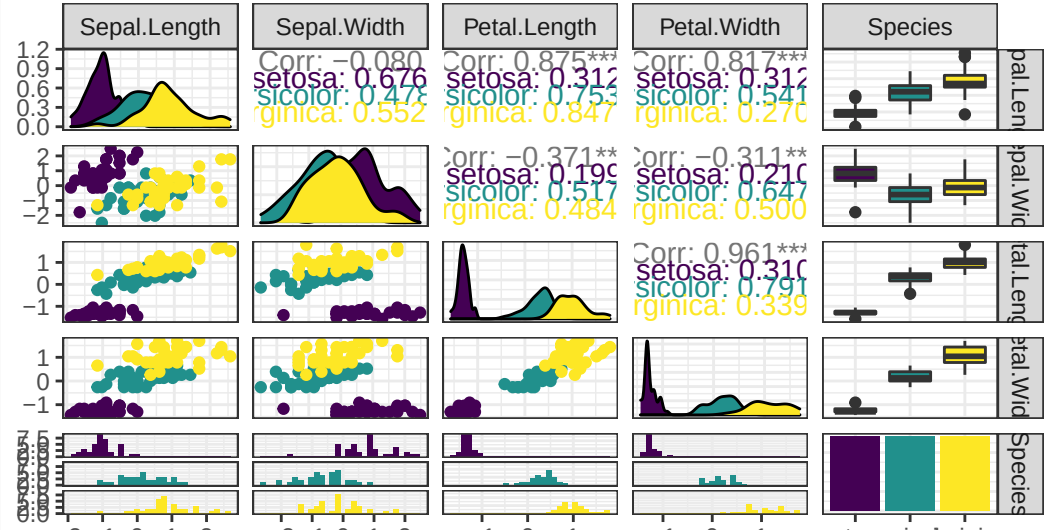
Exemplo


Petal.Length
Petal.Width
Sepal.Length
Sepal.Width
count
value
0
10
20
−2
−1
0
1
2
Species
setosa
versicolor
virginica

Exemplo

```
> ggpairs(iris_treino_t,
+         aes(colour = Species)) +
+   scale_colour_viridis_d() +
+   scale_fill_viridis_d()
```

Exemplo


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species
Corr: −0.080
setosa: 0.676
Corr: 0.875***
setosa: 0.312
Corr: 0.817***
setosa: 0.312
Corr: −0.371**
setosa: 0.199
Corr: −0.311**
setosa: 0.210
Corr: 0.961***
setosa: 0.310
setosa
versicolor
virginica

Exemplo

```
> # preparar o conjunto de teste
>
> iris_teste_t <- bake(iris_rec,
+                      new_data = iris_teste)
>
> # definicao do modelo
>
> iris_knn <-
+   nearest_neighbor() %>%
+   set_engine("kknn") %>%
+   set_mode("classification")
```

Exemplo

```
> # criar workflow
>
> iris_wflow <-
+   workflow() %>%
+   add_recipe(iris_rec) %>%
+   add_model(iris_knn)
>
> # ajuste do modelo
>
> iris_knn_fit <- fit(iris_wflow, iris_treino_t)
```

Exemplo

```
> # validacao cruzada
>
> set.seed(4325)
>
> iris_treino_cv <- vfold_cv(iris_treino_t, v = 5)
```

Exemplo

```
> # criar workflow
>
> iris_wflow_cv <-
+   workflow() %>%
+   add_model(iris_knn) %>%
+   add_formula(Species ~ .)
>
> # rodando validacao cruzada
>
> iris_knn_fit_cv <-
+   iris_wflow_cv %>%
+   fit_resamples(iris_treino_cv)

##
## Attaching package: 'rlang'
```

Exemplo

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.948     5 0.0163  Preprocessor1_Mod~
## 2 roc_auc  hand_till  0.996     5 0.00237 Preprocessor1_Mod~
```

Exemplo

```
> # resultado final
>
> iris_knn_fit

## == Workflow [trained] ==========================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------
## 3 Recipe Steps
##
## * step_downsample()
## * step_center()
## * step_scale()
##
```

Exemplo

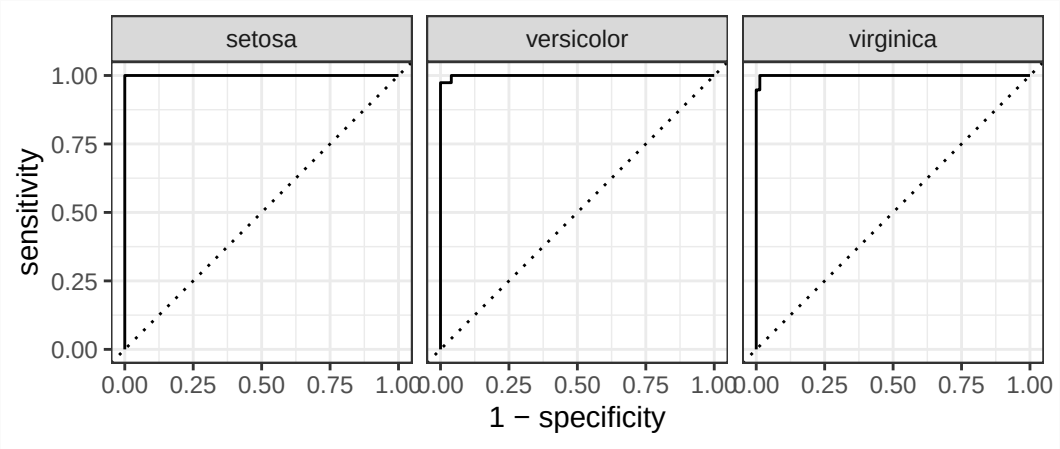
```
> iris_knn_fit %>%
+   predict(iris_teste_t) %>%
+   bind_cols(iris_teste_t) %>%
+   metrics(truth = Species, estimate = .pred_class)

## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.944
## 2 kap      multiclass     0.917
```

Exemplo

```
> iris_knn_fit %>%
+   # probabilidade para cada classe
+   predict(iris_treino_t, type = "prob") %>%
+   bind_cols(iris_treino_t) %>%
+   # grafico
+   roc_curve(Species, .pred_setosa:.pred_virginica) %>%
+   autoplot()
```

Exemplo


setosa
versicolor
virginica
sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 − specificity

Tunning

Tunning

- Pergunta: quais valores de k utilizamos na análise?
- O melhor resultado foi obtido quando $k = 5$
- Será que obteríamos predições melhores com outros valores de k?

Tunning

- Distância euclidiana: sejam $X = \mathbb{R}^n$ e sejam $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ tais que $\boldsymbol{x} = (x_1, \cdots, x_n)$ e $\boldsymbol{y} = (y_1, \cdots, y_n)$. Assim,

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} = \left[\sum_{i=1}^{n} (x_i - y_i)^2 \right]^{\frac{1}{2}}$$

Tunning

- Distância de Manhattan:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

- Distância de Minkowski (ou norma-p):

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left[\sum_{i=1}^{n} |x_i - y_i|^p \right]^{\frac{1}{p}}$$

Tunning

- Como escolher a distância a ser usada?
- A distância euclidiana funciona muito bem em casos lineares, mas é sensível a outliers
- A distância Manhattan funciona melhor em casos com muitas observações aberrantes ou num espaço com muitas dimensões
- Outras distâncias podem ser usadas em outros casos

Tunning

- Queremos encontrar a melhor combinação de valores de k e distância para o problema que estamos analisando
- Ou seja, queremos maximizar o valor da acurácia (ou de alguma outra medida) considerando diferentes valores para k e para as distâncias
- Valores como k e as distâncias, que ajudam na procura do melhor modelo para os nossos dados, são chamados de hiperparâmetros

Tunning

- Vamos ver como encontrar valores próximos dos ideais para os hiperparâmetros do *k*NN através de uma grade de procura
- Iremos definir valores específicos para k e para os tipos de distâncias e ajustaremos modelos para todas as combinações possíveis para os hiperparâmetros
- Esse processo é computacionalmente intenso e seu tempo de execução dependerá diretamente de características como a quantidade de combinações de hiperparâmetros, o tamanho do conjunto de dados e a complexidade da validação cruzada, dentre outros

Exemplo

```
> # definicao do tuning
>
> iris_knn_tune <-
+   nearest_neighbor(
+     neighbors = tune(),
+     weight_func = tune(),
+     dist_power = tune()
+   ) %>%
+   set_engine("kknn") %>%
+   set_mode("classification")
```

Exemplo

```
> iris_knn_tune

## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = tune()
##   weight_func = tune()
##   dist_power = tune()
##
## Computational engine: kknn
```

Exemplo

```
> # grid de procura
>
> iris_knn_grid <- grid_regular(neighbors(range = c(3, 45)),
+                               weight_func(),
+                               dist_power(),
+                               levels = c(22, 2, 2))
```

Exemplo

```
> iris_knn_grid

## # A tibble: 88 x 3
##    neighbors weight_func dist_power
##        <int> <chr>            <dbl>
##  1         3 rectangular          1
##  2         5 rectangular          1
##  3         7 rectangular          1
##  4         9 rectangular          1
##  5        11 rectangular          1
##  6        13 rectangular          1
##  7        15 rectangular          1
##  8        17 rectangular          1
##  9        19 rectangular          1
## 10        21 rectangular          1
```

Exemplo

```
> # workflow
>
> iris_knn_tune_wflow <-
+   workflow() %>%
+   add_model(iris_knn_tune) %>%
+   add_formula(Species ~ .)
```

Exemplo

```
> # ajuste do modelo
>
> iris_knn_fit_tune <-
+   iris_knn_tune_wflow %>%
+   tune_grid(
+     resamples = iris_treino_cv,
+     grid = iris_knn_grid
+   )
```

Exemplo

```
> iris_knn_fit_tune

## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits          id    .metrics          .notes
##   <list>          <chr> <list>            <list>
## 1 <split [91/23]> Fold1 <tibble [72 x 7]> <tibble [0 x 1]>
## 2 <split [91/23]> Fold2 <tibble [72 x 7]> <tibble [0 x 1]>
## 3 <split [91/23]> Fold3 <tibble [72 x 7]> <tibble [0 x 1]>
## 4 <split [91/23]> Fold4 <tibble [72 x 7]> <tibble [0 x 1]>
## 5 <split [92/22]> Fold5 <tibble [72 x 7]> <tibble [0 x 1]>
```

Exemplo

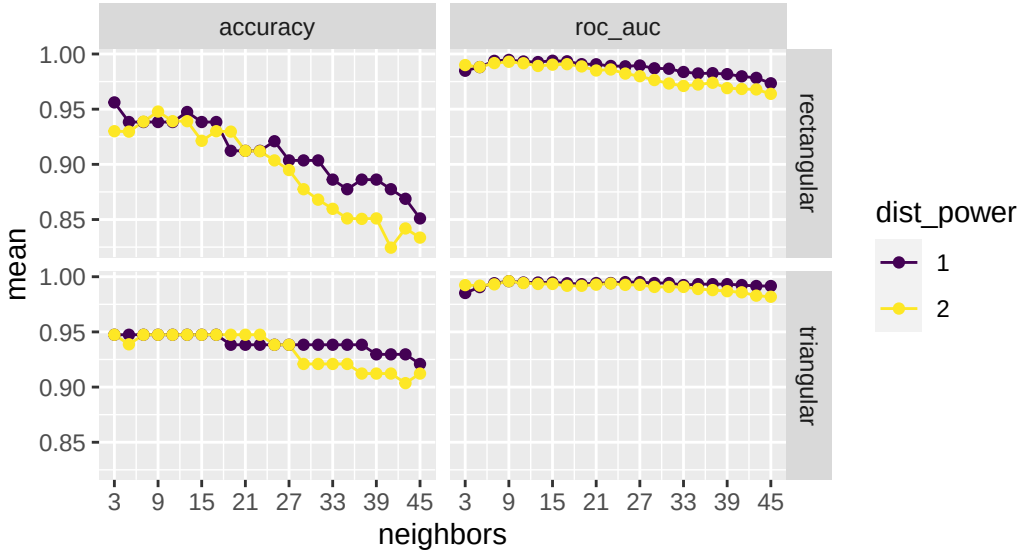
```
> # resultados
>
> collect_metrics(iris_knn_fit_tune)

## # A tibble: 176 x 9
##    neighbors weight_func dist_power .metric .estimator  mean
##        <int> <chr>            <dbl> <chr>   <chr>      <dbl>
##  1         3 rectangular          1 accura~ multiclass 0.956
##  2         3 rectangular          1 roc_auc hand_till  0.985
##  3         5 rectangular          1 accura~ multiclass 0.938
##  4         5 rectangular          1 roc_auc hand_till  0.988
##  5         7 rectangular          1 accura~ multiclass 0.938
##  6         7 rectangular          1 roc_auc hand_till  0.994
##  7         9 rectangular          1 accura~ multiclass 0.938
##  8         9 rectangular          1 roc_auc hand_till  0.995
```

Exemplo

```
> iris_knn_fit_tune %>%
+   collect_metrics() %>%
+   mutate(dist_power = factor(dist_power)) %>%
+   ggplot(., aes(x = neighbors, y = mean, color = dist_power)) +
+   geom_line() +
+   geom_point() +
+   facet_grid(weight_func ~ .metric) +
+   scale_x_continuous(breaks = seq(3, 45, 6)) +
+   scale_colour_viridis_d()
```

Exemplo


accuracy
roc_auc
rectangular
triangular
mean
1.00
0.95
0.90
0.85
3
9
15
21
27
33
39
45
neighbors
dist_power
1
2

Exemplo

```
> # melhores modelos
>
> iris_knn_fit_tune %>%
+   show_best("roc_auc")

## # A tibble: 5 x 9
##   neighbors weight_func dist_power .metric .estimator  mean
##       <int> <chr>            <dbl> <chr>   <chr>      <dbl>
## 1         9 triangular           1 roc_auc hand_till  0.996
## 2         9 triangular           2 roc_auc hand_till  0.996
## 3        25 triangular           1 roc_auc hand_till  0.995
## 4        27 triangular           1 roc_auc hand_till  0.995
## 5        11 triangular           1 roc_auc hand_till  0.995
## # ... with 3 more variables: n <int>, std_err <dbl>,
## #   .config <chr>
```

Exemplo

```
> # melhores modelos
>
> iris_knn_fit_tune %>%
+   show_best("accuracy")

## # A tibble: 5 x 9
##   neighbors weight_func dist_power .metric .estimator  mean
##       <int> <chr>            <dbl> <chr>   <chr>      <dbl>
## 1         3 rectangular          1 accura~ multiclass 0.956
## 2         9 rectangular          2 accura~ multiclass 0.948
## 3        13 rectangular          1 accura~ multiclass 0.947
## 4         3 triangular           1 accura~ multiclass 0.947
## 5         5 triangular           1 accura~ multiclass 0.947
## # ... with 3 more variables: n <int>, std_err <dbl>,
## #   .config <chr>
```

Exemplo

```
> # melhor modelo
>
> iris_knn_best <-
+   iris_knn_fit_tune %>%
+   select_best("accuracy")
>
> iris_knn_final <-
+   iris_knn_tune_wflow %>%
+   finalize_workflow(iris_knn_best)
>
> iris_knn_final <- fit(iris_knn_final, iris_treino_t)
```

Exemplo

```
> iris_knn_final

## == Workflow [trained] ==========================================
## Preprocessor: Formula
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------
## Species ~ .
##
## -- Model -----------------------------------------------------
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(3L,
##
## Type of response variable: nominal
```

Exemplo

```
> # resultados no conjunto de teste
>
> resultado <-
+   iris_teste_t %>%
+   bind_cols(predict(iris_knn_final, iris_teste_t) %>%
+               rename(predicao_knn = .pred_class))
```

Exemplo

```
> # resultados no conjunto de teste
>
> metrics(resultado,
+         truth = Species,
+         estimate = predicao_knn)

## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.944
## 2 kap      multiclass     0.917
```

Exercícios

Exercícios

O pacote `mlbench` possui um conjunto de dados chamado `Sonar`. São 208 observações de objetos cilíndricos de metal (M) ou objetos rochosos aproximadamente cilíndricos (R). As suas 60 variáveis preditoras são referentes à energia do sonar em bandas de frequência diferentes.

Nosso objetivo é utilizar o algoritmo *k*NN para modelar esses dados, construindo um algoritmo de classificação capaz de distinguir entre as classes M e R.

Exercícios

1. Carregue o conjunto de dados na memória através dos comandos

   ```
   > library(mlbench)
   > data(Sonar)
   ```

   A PCA desse conjunto de dados sugere que os grupos são linearmente separáveis? Por quê?
2. Separe 70% dos seus dados para o conjunto de treinamento.
3. Crie um modelo de predição para esse conjunto de dados. Qual é a acurácia do modelo e o melhor valor de k, utilizando validação cruzada com 7 grupos?
4. O seu modelo está bem ajustado ao dados ou houve *overfitting*? Justifique.
5. Visualize o resultado da sua classificação usando uma curva ROC.

k Vizinhos Mais Próximos

EST0133 - Introdução à Modelagem de Big Data

Marcus Nunes
https://introbigdata.org/
https://marcusnunes.me/

Universidade Federal do Rio Grande do Norte