Análise de Dados Aplicada à Computação

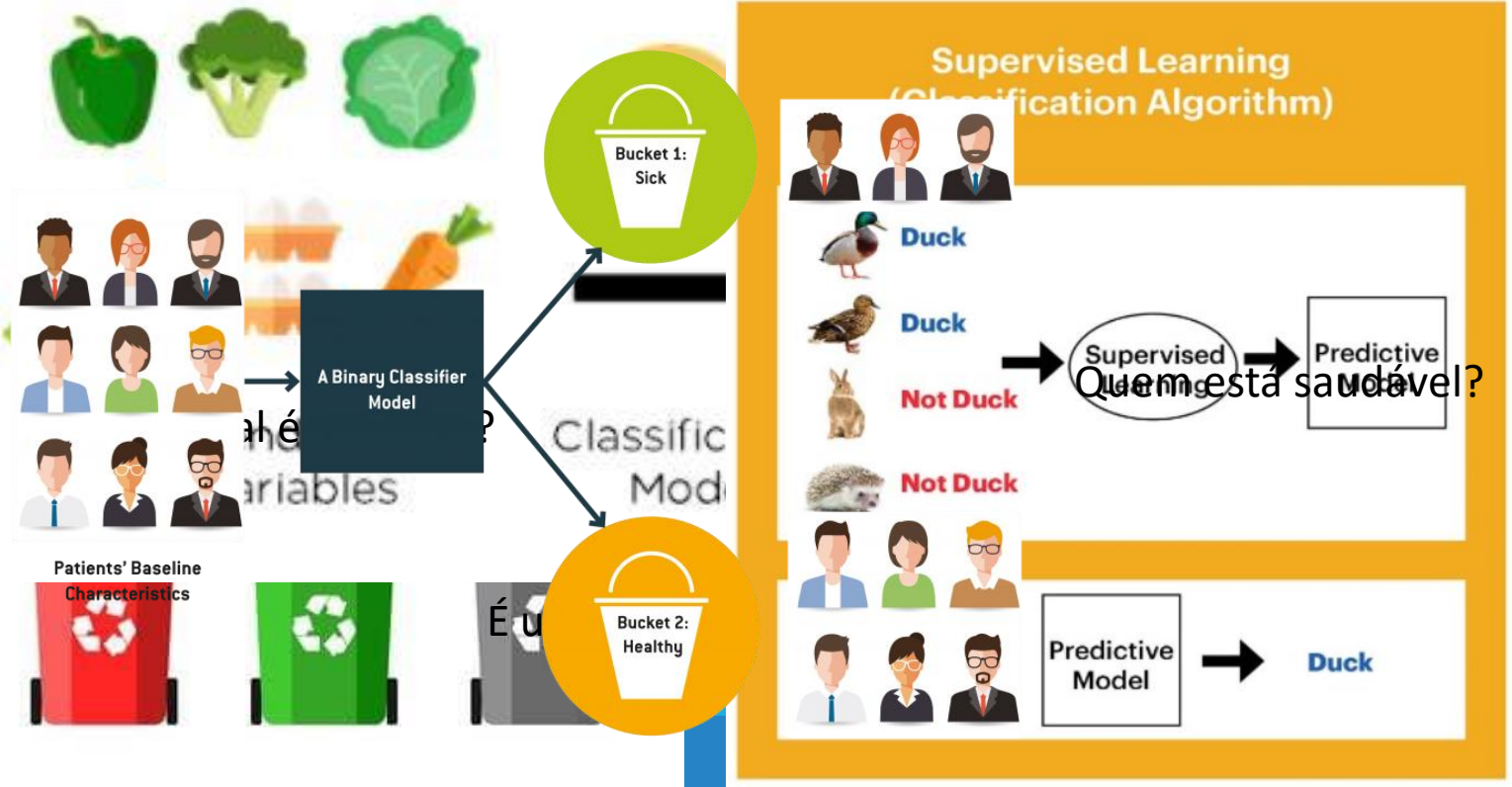
CLASSIFICAÇÃO

Prof. M.Sc. Howard Roatti

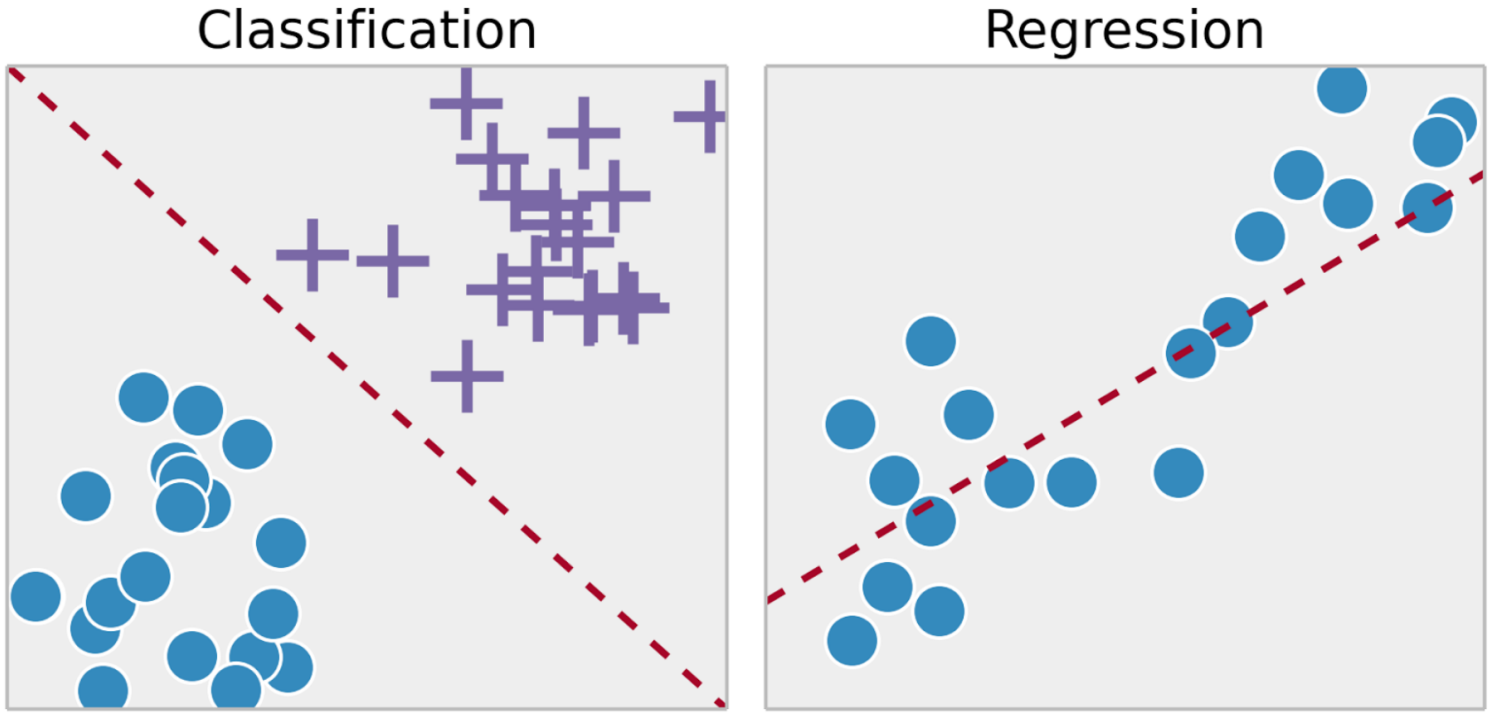
Sumário

1. Definição de Classificação
2. Naive Bayes
3. Regressão Logística
4. Avaliação de Modelos de Classificação

Classificação



Classificação



https://medium.com/machina-sapiens/algoritmos-de-aprendizagem-de-m%C3%A1quina-qual-deles-escolher-67040ad68737

Naive Bayes

- Baseado no **Teorema de Bayes** (saiba mais) que computa a probabilidade de um evento ocorrer, observando o acontecimento de outro evento associado
- Tem como objetivo calcular ***probabilidades condicionais*** do tipo:
  - P(**Umidade Alta**|**Chover**)
  - P(**Alta Velocidade**|**Acidente de Carro**)
  - P(**Jogou na Mega-Sena**|**Enriquecer**)

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- É chamada de **Naive** (Ingênuo) por assumir que os eventos são independentes, permitindo calcular a probabilidade através do método da Máxima Probabilidade

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Naive Bayes

- Passos:
  1. Separe o conjunto de dados por classes
  2. Calcule a média e o desvio padrão de todos os atributos por classes
  3. Para cada registro que queira classificar, calcule o produtório das probabilidades dos atributos ocorrerem para cada classe analisada
  4. A classe que obtiver a maior probabilidade será a classe associada ao registro ainda não visto.

Naive Bayes

- Python

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB

loan_data = pd.read_csv('./DataSets/loan_data.csv.gz')

# convert to categorical
loan_data.outcome = loan_data.outcome.astype('category')
loan_data.outcome.cat.reorder_categories(['paid off', 'default'])
loan_data.purpose_ = loan_data.purpose_.astype('category')
loan_data.home_ = loan_data.home_.astype('category')
loan_data.emp_len_ = loan_data.emp_len_.astype('category')

predictors = ['purpose_', 'home_', 'emp_len_']
outcome = 'outcome'
X = pd.get_dummies(loan_data[predictors], prefix='', prefix_sep='')
y = loan_data[outcome]
```

Naive Bayes

- Python

```
naive_model = MultinomialNB(alpha=0.01, fit_prior=True)
naive_model.fit(X, y)
```

```
MultinomialNB(alpha=0.01, class_prior=None, fit_prior=True)
```

```
new_loan = X.loc[146:146, :]
print('predicted class: ', naive_model.predict(new_loan)[0])

probabilities = pd.DataFrame(naive_model.predict_proba(new_loan),
                             columns=naive_model.classes_)
print('predicted probabilities',)
probabilities
```

```
predicted class:  default
predicted probabilities
```

	default	paid off
0	0.653696	0.346304

Regressão Logística

- A regressão logística é semelhante à regressão linear múltipla, exceto pelo seu resultado ser binário (0 ou 1).
- Possui um desempenho muito bom em relação a outros modelos no quesito velocidade
- Utiliza a função logística para obter o resultado entre 0 e 1, ajustado pela função logit

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}}$$

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Regressão Logística

- Python

```
from sklearn.linear_model import LogisticRegression

predictors = ['purpose_', 'home_', 'emp_len_']

outcome = 'outcome'
X = pd.get_dummies(loan_data[predictors], prefix='', prefix_sep='',
                   drop_first=True)
y = loan_data[outcome]

logit_reg = LogisticRegression(penalty='l2', C=1e42, solver='liblinear')
logit_reg.fit(X, y)
```

```
LogisticRegression(C=1e+42, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Regressão Logística

- Python

```
new_loan = X.loc[146:146, :]

print('predicted class: ', logit_reg.predict(new_loan)[0])

probabilities = pd.DataFrame(logit_reg.predict_proba(new_loan),
                             columns=logit_reg.classes_)
print()
print('predicted probabilities: ')
probabilities
```

```
predicted class:  default

predicted probabilities:
```

	default	paid off
0	0.658086	0.341914

Regressão Logística

- Python

```
print('intercept ', logit_reg.intercept_[0])
print('classes', logit_reg.classes_)
pd.DataFrame({'coeff': logit_reg.coef_[0]},
             index=X.columns)
```

```
intercept  -0.10330239562885249
classes ['default' 'paid off']
```

	coeff
debt_consolidation	-0.260492
home_improvement	-0.106634
major_purchase	0.157170
medical	-0.204026
other	-0.329056
small_business	-1.012170
OWN	-0.138570
RENT	-0.240933
> 1 Year	0.460696

```
print(loan_data['purpose_'].cat.categories)
print(loan_data['home_'].cat.categories)
print(loan_data['emp_len_'].cat.categories)
```

```
Index(['credit_card', 'debt_consolidation', 'home_improvement',
       'major_purchase', 'medical', 'other', 'small_business'],
      dtype='object')

Index(['MORTGAGE', 'OWN', 'RENT'], dtype='object')

Index([' < 1 Year', ' > 1 Year'], dtype='object')
```

Métricas de Avaliação

- Para determinar o quão bom um classificador é, faz necessário a utilização de métricas que permitam medir a qualidade de seus resultados.
- Discutiremos: **(1)** Acurácia e Erro **(2)** Precision e Recall **(3)** Medida-F e F1 **(4)** Curva ROC **(5)** AUC **(6)** Validação Cruzada

Métricas de Avaliação

- Tabela de Contingência:

$Case$	$\mathcal{T}(d_j, c_p) = 1$	$\mathcal{T}(d_j, c_p) = 0$	$Total$
$\mathcal{F}(d_j, c_p) = 1$	$n_{f,t}$	$n_f - n_{f,t}$	n_f
$\mathcal{F}(d_j, c_p) = 0$	$n_t - n_{f,t}$	$N_t - n_f - n_t + n_{f,t}$	$N_t - n_f$
$All\ docs$	n_t	$N_t - n_t$	N_t

- $n_{f,t}$: número de documentos que ambas funções de treinamento e do classificador associaram a classe c_p
- $n_t - n_{f,t}$: número de documentos de treino na classe c_p que foram mal classificados

Métricas de Avaliação

- **Tabela de Contigência → Matriz de Confusão**

		Não
Real	**Sim**	so Negativo (FN)
	Não	deiro Negativo (VN)

Métricas de Avaliação

- **Acurácia:** contabiliza a proporção de previsões correta

$$accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

- **Erro:** contabiliza a proporção de previsões incorretas

$$error = 1 - accuracy$$

		Detectada	
		Sim	**Não**
Real	**Sim**	Verdadeiro Positivo (VP)	Falso Negativo (FN)
	Não	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Métricas de Avaliação

- **Precision:** proporção de amostras classificadas corretamente como **Sim** e realmente são **Sim**
- **Recall:** proporção de amostras que são **Sim** e que foram classificadas como **Sim**
- **Specificty:** proporção de amostras que são **Não** e que foram classificados como **Não**

		Detectada	
		Sim	**Não**
Real	**Sim**	Verdadeiro Positivo (VP)	Falso Negativo (FN)
	Não	Falso Positivo (FP)	Verdadeiro Negativo (VN)

$$precision = \frac{VP}{VP + FP} \quad recall = \frac{VP}{VP + FN} \quad specificity = \frac{VN}{VN + FP}$$

Métricas de Avaliação

measure		calculated value
Error rate	ERR	6 / 20 = 0.3
Accuracy	ACC	14 / 20 = 0.7
Sensitivity True positive rate Recall	SN TPR REC	6 / 10 = 0.6
Specificity True negative rate	SP TNR	8 / 10 = 0.8
Precision Positive predictive value	PREC PPV	6 / 8 =0.75

Métricas de Avaliação

- **F1:** é a média harmônica entre o ***precision*** e o ***recall***

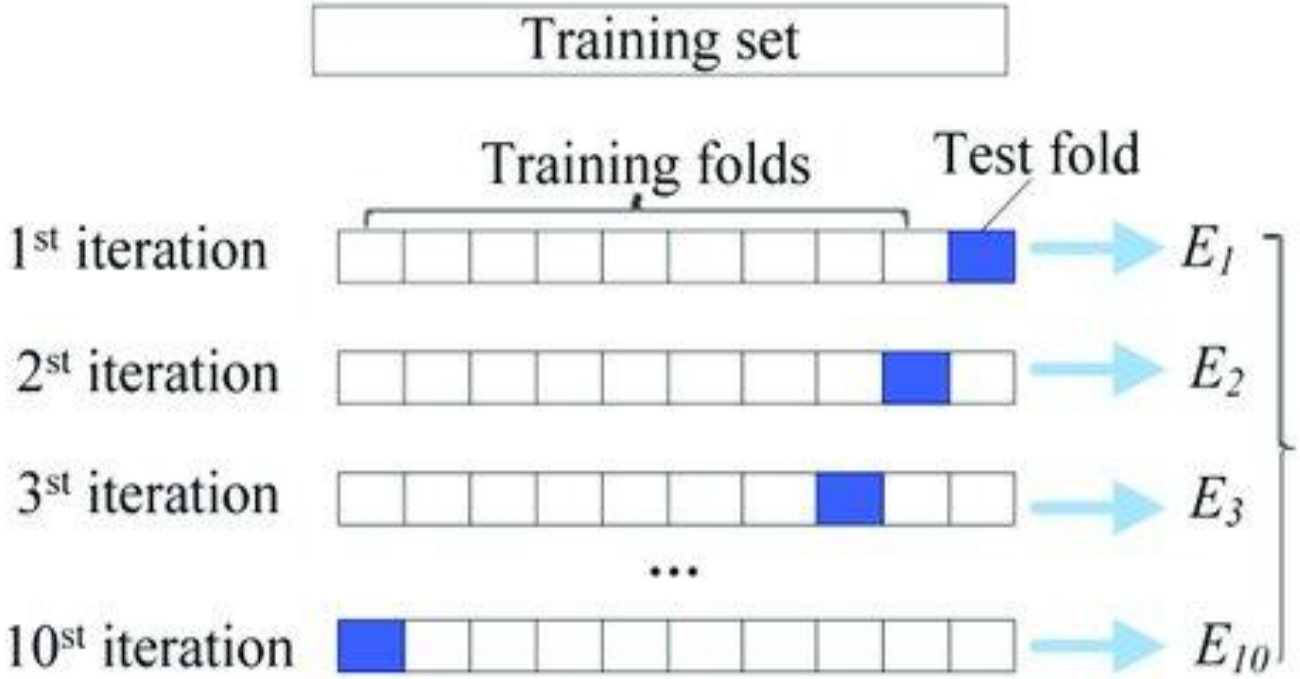
$$F_1 = \frac{2PR}{P + R}$$

- ***Micro-F1:*** $micro - F_1 = \frac{2PR}{P + R}$ $P = \sum_{c_p \in C} precision(c_p)$ $R = \sum_{c_p \in C} recall(c_p)$
- ***Macro-F1:*** $macro - F_1 = \frac{\sum_{p=1}^{|C|} F_1(c_p)}{|C|}$

Métricas de Avaliação

- ***Cross-validation* (Validação Cruzada):** métodos comuns que garantem a validação estatística dos resultados constrói diferentes modelos de classificação, para isso, divide o conjuto de treino D_t em k conjuntos disjuntos (*folds*)
- Um classificador treina com o conjunto de treino menos o i-ésimo *fold* e testa com o i-ésimo *fold*.
- Cada classificador é avaliado de forma independente usando *Recall*, *Precision* e F1.
- A validação cruzada calcula a média das k medidas
- É comum adotar k = 10, esse método é chamado de *ten-fold cross-validation*

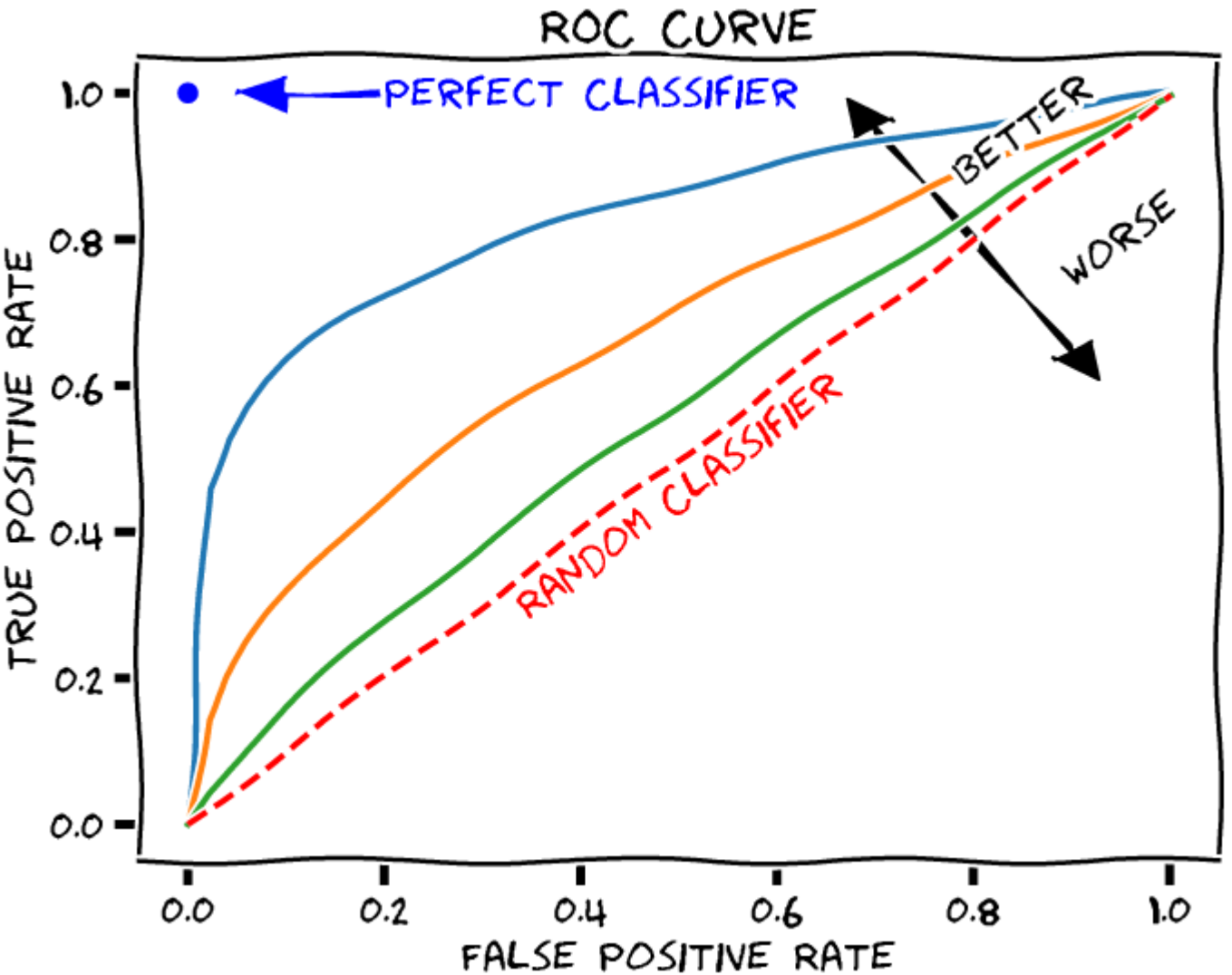
Métricas de Avaliação



$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

M

- **C**
plot
  - (
    p
  - (
    r
- A
qua
sup

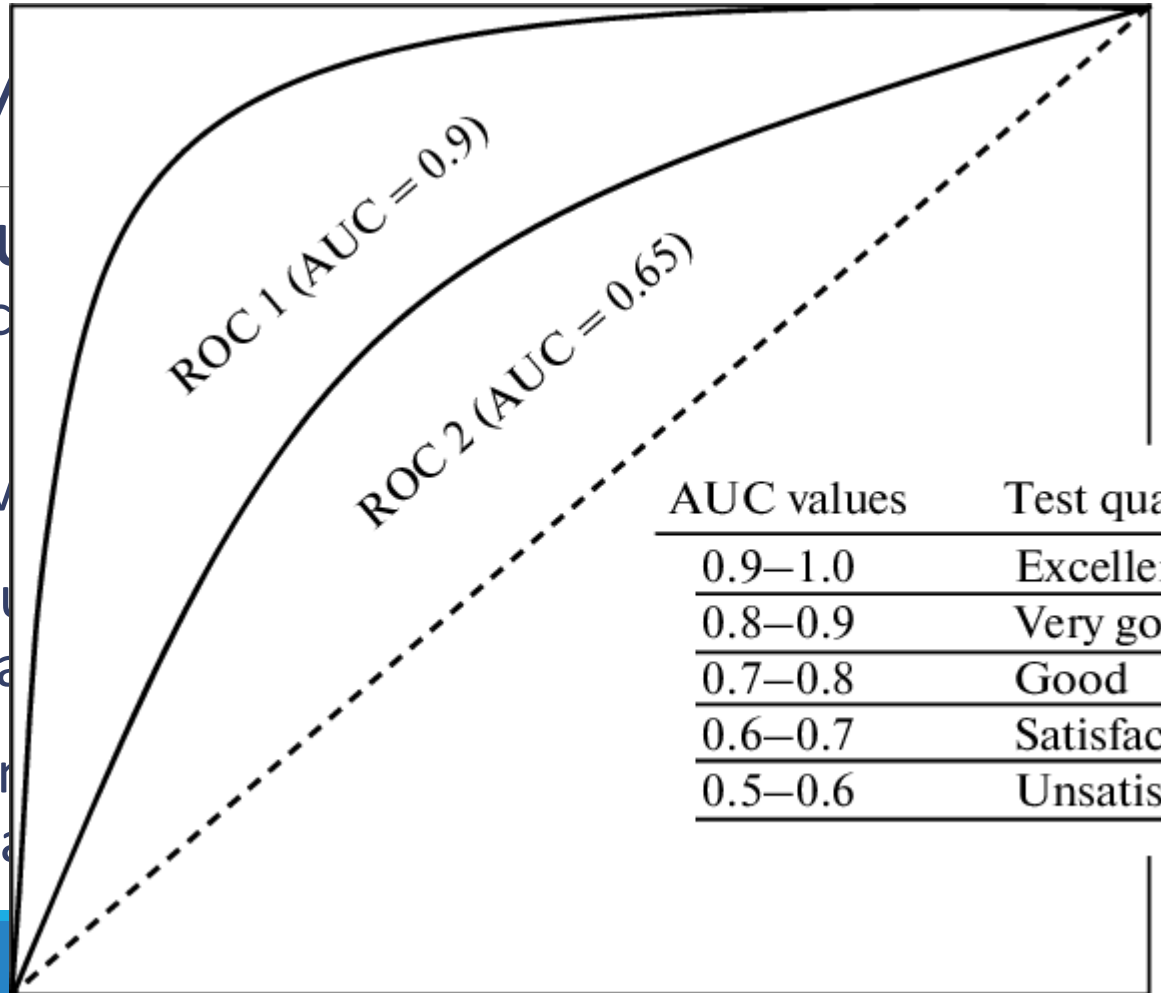


is

lo

M

- **AU** … uma única
- A … curv…
- Qu… ador. Uma…
- Un… a linha…



AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

Métricas de Avaliação

- **Python**

```
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support
from sklearn.metrics import roc_curve, accuracy_score, roc_auc_score

# Confusion matrix
pred = logit_reg.predict(X)
pred_y = logit_reg.predict(X) == 'default'
true_y = y == 'default'
true_pos = true_y & pred_y
true_neg = ~true_y & ~pred_y
false_pos = ~true_y & pred_y
false_neg = true_y & ~pred_y

conf_mat = pd.DataFrame([[np.sum(true_pos), np.sum(false_neg)], [np.sum(false_pos), np.sum(true_neg)]],
                        index=['Y = default', 'Y = paid off'],
                        columns=['Yhat = default', 'Yhat = paid off'])
print(conf_mat)
```

	Yhat = default	Yhat = paid off
Y = default	14337	8334
Y = paid off	8149	14522

Métricas de Avaliação

- **Python**

```
conf_mat = confusion_matrix(y, logit_reg.predict(X))
print('Precision', conf_mat[0, 0] / sum(conf_mat[:, 0]))
print('Recall', conf_mat[0, 0] / sum(conf_mat[0, :]))
print('Specificity', conf_mat[1, 1] / sum(conf_mat[1, :]))
```

```
Precision 0.6375967268522637
Recall 0.6323938070662961
Specificity 0.640554011733051
```

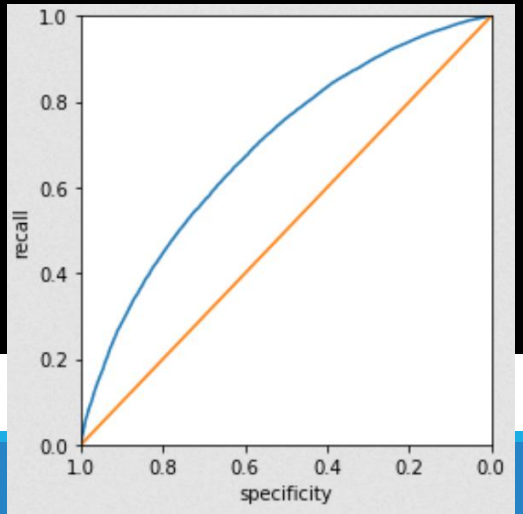
Métricas de Avaliação

- **Python**

```
fpr, tpr, thresholds = roc_curve(y, logit_reg.predict_proba(X)[:, 0],
                                 pos_label='default')
roc_df = pd.DataFrame({'recall': tpr, 'specificity': 1 - fpr})

ax = roc_df.plot(x='specificity', y='recall', figsize=(4, 4), legend=False)
ax.set_ylim(0, 1)
ax.set_xlim(1, 0)
ax.plot((1, 0), (0, 1))
ax.set_xlabel('specificity')
ax.set_ylabel('recall')


plt.tight_layout()
plt.show()
```



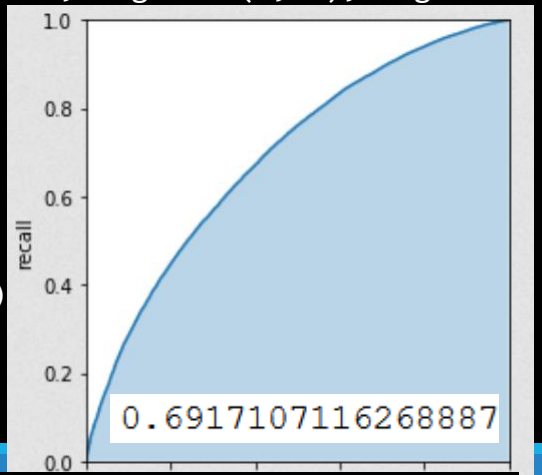
Métricas de Avaliação

- **Python**

```
fpr, tpr, thresholds = roc_curve(y, logit_reg.predict_proba(X)[:,0],
                                 pos_label='default')
roc_df = pd.DataFrame({'recall': tpr, 'specificity': 1 - fpr})

ax = roc_df.plot(x='specificity', y='recall', figsize=(4, 4), legend=False)
ax.set_ylim(0, 1)
ax.set_xlim(1, 0)
# ax.plot((1, 0), (0, 1))
ax.set_xlabel('specificity')
ax.set_ylabel('recall')
ax.fill_between(roc_df.specificity,
                0,
                roc_df.recall, alpha=0.3)

plt.tight_layout()
plt.show()
```




```
print(np.sum(roc_df.recall[:-1] * np.diff(1 - roc_df.specificity)))
```

Referências

- Bruce, P.; Bruce, A.; **Estatística Prática para Cientista de Dados**: 50 Conceitos Essenciais; Rio de Janeiro; Alta Books; 2019.
- Morettin, P. A.; Bussab, W. O.; **Estatística Básica**. 8 ed. São Paulo: Saraiva, 2013.
- https://www.s-cubed-global.com/biometrics/statistics-and-machine-learning
- https://intellipaat.com/blog/tutorial/machine-learning-tutorial/classification-machine-learning/
- https://www.researchgate.net/publication/276079439_Metabolic_profiling_of_human_blood/figures
- https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- https://classeval.wordpress.com/introduction/basic-evaluation-measures/

Referências

- https://scikit-learn.org/stable/modules/cross_validation.html
- https://www.researchgate.net/publication/326465007_RFAmyloid_A_Web_Server_for_Predicting_Amyloid_Proteins/figures?lo=1

FAESA
CENTRO UNIVERSITÁRIO

Análise de Dados Aplicada à Computação

PROF. M.SC HOWARD ROATTI