

**2025-04-20**

referencia de dados e conteúdo e abstrações de juan perri

**referencia para o inicio do projeto e metricas foi o artigo de tcc**

no começo foi pensado fazer o projeto de forma linear mas com o passar do tempo tivermos ideia de usar um llm local com esses dados para automatizar e conseguimos gerar código se baseando nessas documentações e tbm fazendo a configuração de hiper parâmetros, logo essa ideia foi descartada por falta de tempo mas futuramente pode ser interessante tivemos varias reuniões onde fizemos brainstorm que lemos e trabalhamos com esse conteúdo, o uso de prompts foram feitos usando o aprendizado reforçado e usamos o gpt o1, o4mini, claude sonnet, gemini 2.5 pro para aprender a como fazer plots e tirar duvidas da implementações de alguns algoritmos, bem pensamos deixar o código com padrões de software mas o tempo e o código se tornaria mais complexo tbm, então fizemos uma abordagem mais direta.

IAGO MARTINS BOUCINHA

**Um modelo de previsão de resultados de  
futebol utilizando Machine Learning**

Monografia apresentada como requisito parcial  
para a obtenção do grau de Bacharel em  
Engenharia da Computação

Orientador: Prof. Dra. Renata Galante

- 
- [https://scikit-learn.org/stable/modules/model\\_evaluation.html#regression-metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics)  
[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)  
<https://gianmedeirao.medium.com/como-fazer-labelencoder-em-um-dataframe-python-sklearn-655ba2c6ae7e>  
[https://medium.com/data-and-beyond/voting-regressor-intuition-and-implementation-0359771b5204!\[\[Captura de tela 2025-04-20 124502.png\]\]](https://medium.com/data-and-beyond/voting-regressor-intuition-and-implementation-0359771b5204![[Captura de tela 2025-04-20 124502.png]])  
[https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_likelihood\\_ratios.html#sphx-glr-auto-examples-model-selection-plot-likelihood-ratios-py](https://scikit-learn.org/stable/auto_examples/model_selection/plot_likelihood_ratios.html#sphx-glr-auto-examples-model-selection-plot-likelihood-ratios-py)  
[https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_cv\\_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py](https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py)  
<https://medium.com/@edubrazrabello/cross-validation-avaliando-seu-modelo-de-machine-learning-1fb70df15b78>  
<https://medium.com/@nandiniverma78988/understanding-k-nearest-neighbors-knn-regression-in-machine-learning-c751a7cf516c>

[https://www.youtube.com/watch?v=kA-P9ood-cE&t=551s&ab\\_channel=OpenAI](https://www.youtube.com/watch?v=kA-P9ood-cE&t=551s&ab_channel=OpenAI)

<https://www.datacamp.com/pt/tutorial/auc>

<https://medium.com/@mateuspdua/machine-learning-m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-e-recall-d44c72307959>

[https://scikit-learn.org/stable/modules/generated/sklearn.gaussian\\_process.GaussianProcessClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html)

[https://scikit-learn.org/stable/auto\\_examples/gaussian\\_process/plot\\_gpc\\_xor.html#sphx-glr-auto-examples-gaussian-process-plot-gpc-xor-py](https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc_xor.html#sphx-glr-auto-examples-gaussian-process-plot-gpc-xor-py)

[https://scikit-learn.org/stable/auto\\_examples/gaussian\\_process/plot\\_gpc\\_iris.html#sphx-glr-auto-examples-gaussian-process-plot-gpc-iris-py](https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc_iris.html#sphx-glr-auto-examples-gaussian-process-plot-gpc-iris-py)

[https://scikit-learn.org/stable/supervised\\_learning.html](https://scikit-learn.org/stable/supervised_learning.html)

<https://scikit-learn.org/stable/modules/svm.html>

<https://scikit-learn.org/stable/modules/sgd.html>

<https://scikit-learn.org/stable/modules/neighbors.html>

[https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)

<https://scikit-learn.org/stable/modules/tree.html>

[https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/stable/modules/neural_networks_supervised.html)

<https://www.dio.me/articles/como-usar-python-para-analise-de-dados-introducao>

<https://vidaestudantil.com/podcasts/como-adicionar-figuras-em-latex-cl-6/>

[https://scikit-learn.org/stable/modules/learning\\_curve.html](https://scikit-learn.org/stable/modules/learning_curve.html)

[https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_roc\\_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py](https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py)

[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

[https://scikit-learn.org/stable/modules/grid\\_search.html](https://scikit-learn.org/stable/modules/grid_search.html)

[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)

[https://scikit-learn.org/stable/modules/learning\\_curve.html](https://scikit-learn.org/stable/modules/learning_curve.html)

## Eficiência na execução

A implementação do scikit-learn é  $O(n^4)$ , portanto pode ser difícil escalar para tamanhos maiores. Usar um kernel linear ou o modelo LinearSVC pode melhorar o desempenho da execução, talvez à custa da precisão. Aumentar o valor do parâmetro `cache_size` pode reduzir a ordem para  $O(n^3)$ .

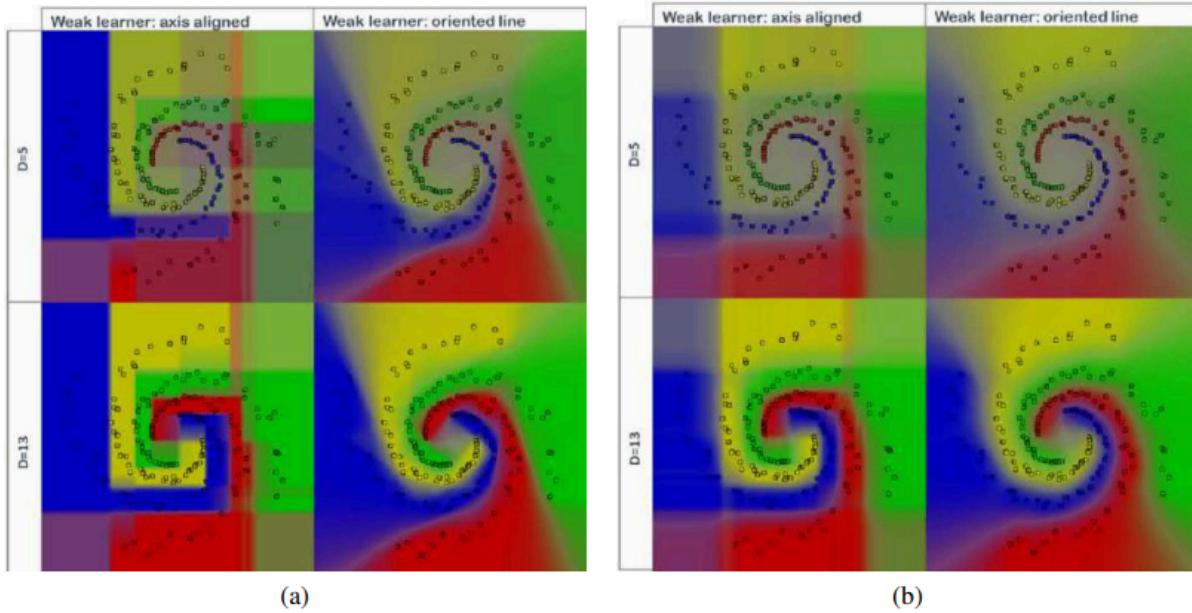
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
CURSO DE ENGENHARIA DE COMPUTAÇÃO

IAGO MARTINS BOUCINHA

**Um modelo de previsão de resultados de  
futebol utilizando Machine Learning**

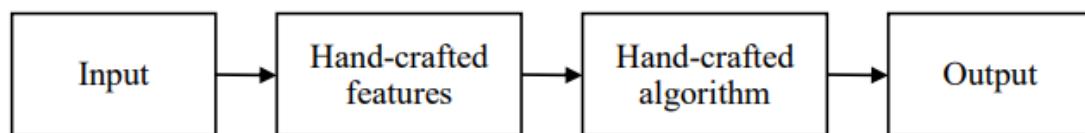
Monografia apresentada como requisito parcial  
para a obtenção do grau de Bacharel em  
Engenharia da Computação

Orientador: Prof. Dra. Renata Galante



**Figure 5.14** Random forest decision surfaces (Criminisi and Shotton 2013) © 2013 Springer. Figures (a) and (b) show smaller and larger amounts of “noise” between the  $T = 400$  tree forests obtained by using  $\rho = 500$  and  $\rho = 5$  random hypotheses at each split node. Within each figure, the two rows show trees of different depths ( $D = 5$  and  $13$ ), while the columns show the effects of using axis-aligned or linear decision surfaces (“weak learners”).

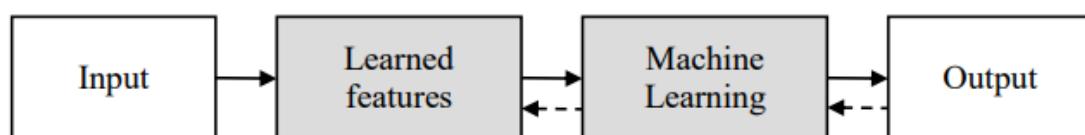
By only looking at a random subset  $\rho$  of all the training examples, each tree ends up having different



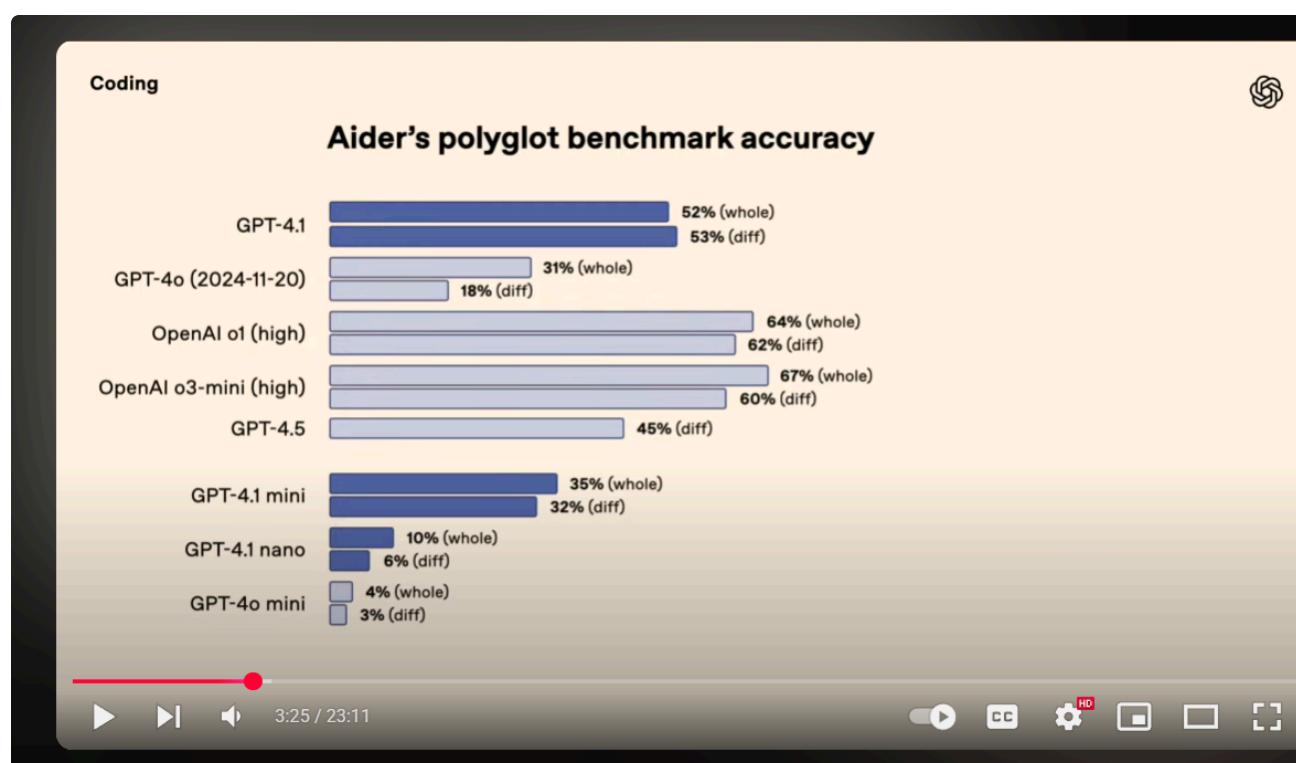
(a) Traditional vision pipeline

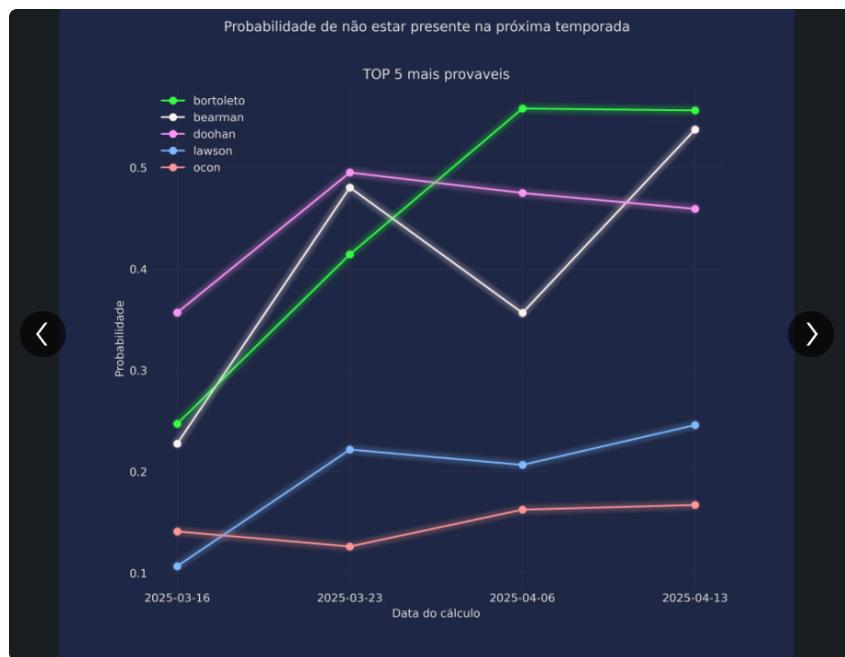


(b) Classic machine learning pipeline



(c) Deep learning pipeline





Téo Calvo • 2º  
Eu transformo vidas por meio do ...  
1 d • 3

ra versao de ML para prever quem continua na proxima temp. da F1.

A ideia é fazer previsões a cada fim de GP, com base nos resultados históricos. Quanto mais alta a prob, pior para o piloto.

Infelizmente a coisa não parece muito boa para nós brasileiros.

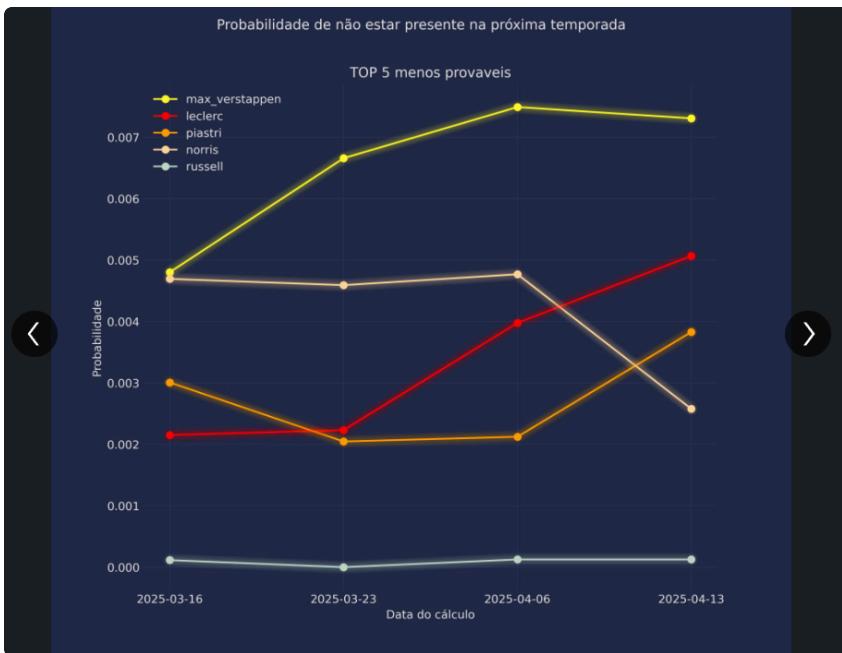
O modelo ainda pode melhorar bastante em relação a metodologia de treinamento. São +60 variáveis e consideramos GPs desde o ano 2000.

Usamos um modelo de Random Forest. Vou deixar nos comentários a lista das principais variáveis.

Meu principal desconforto é com a performance muito alta do modelo:

ACC Treino: 0.9746  
AUC Treino: 0.9970

ACC Test: 0.9366  
AUC Test: 0.9823



Téo Calvo • 2º

Eu transformo vidas por meio do ...

1 d • 5

+ Seguir X

ra versão de IML para prever quem continua na proxima temp. da F1.

A ideia é fazer previsões a cada fim de GP, com base nos resultados históricos. Quanto mais alta a prob, pior para o piloto.

Infelizmente a coisa não parece muito boa para nós brasileiros.

O modelo ainda pode melhorar bastante em relação à metodologia de treinamento. São +60 variáveis e consideraremos GPs desde o ano 2000.

Usamos um modelo de Random Forest. Vou deixar nos comentários a lista das principais variáveis.

Meu principal desconforto é com a performance muito alta do modelo:

ACC Treino: 0.9746

AUC Treino: 0.9970

ACC Test: 0.9366

AUC Test: 0.9823

# Explicando os modelos de regressão

A maioria das técnicas usadas para explicar os modelos de classificação se aplicam também aos modelos de regressão. Neste capítulo, mostrarei como usar a biblioteca SHAP para interpretar os modelos de regressão.

Interpretaremos um modelo XGBoost para o conjunto de dados habitacionais de Boston:

```
>>> import xgboost as xgb  
>>> xgr = xgb.XGBRegressor(  
... random_state=42, base_score=0.5  
... )  
>>> xgr.fit(bos_X_train, bos_y_train)
```

## Shapley

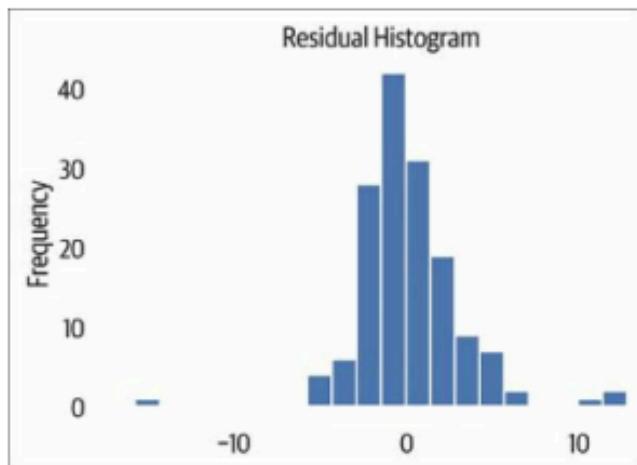
Sou grande fã do Shapley porque ele não depende do modelo. Essa biblioteca também nos dá insights globais sobre o nosso modelo e ajuda a explicar previsões individuais. Caso você tenha um modelo caixa-preta, acho essa biblioteca bastante útil.

Inicialmente, veremos a previsão para o índice 5. Nosso modelo prevê que o valor será 27,26:

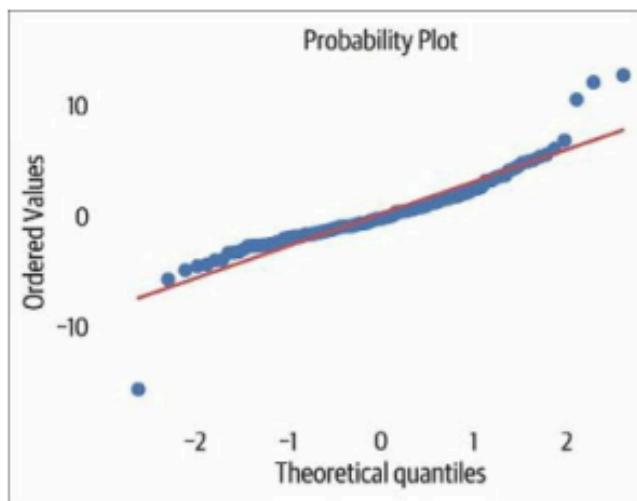
```
>>> sample_idx = 5  
>>> xgr.predict(bos_X.iloc[[sample_idx]])  
array([27.269186], dtype=float32)
```

Para usar o modelo, temos de criar um TreeExplainer a partir de nosso modelo e estimar os valores SHAP para nossas amostras. Se quisermos usar o Jupyter e ter uma interface interativa, também será necessário chamar a função initjs:

```
>>> import shap  
>>> shap.initjs()  
  
>>> exp = shap.TreeExplainer(xgr)  
>>> vals = exp.shap_values(bos_X)
```



*Figura 15.2 – Histograma dos resíduos.*



*Figura 15.3 – Gráfico de probabilidade dos resíduos.*

O teste de Kolmogorov-Smirnov é capaz de avaliar se uma distribuição é normal. Se o valor  $p$  for significativo ( $< 0,05$ ), é sinal de que os valores não apresentam uma distribuição normal.

Esse teste também falha, informando que os resíduos não têm distribuição normal:

```
>>> stats.kstest(resids, cdf="norm")
KstestResult(statistic=0.1962230021010155, pvalue=1.3283596864921421e-05)
```

```
... "p-value",
... "f-value",
... "f p-value",
...
]
>>> for name, num in zip(name, hb):
...     print(f'{name}: {num:.2f}')
Lagrange multiplier statistic: 3.6e+01
p-value: 0.00036
f-value: 3.3
f p-value: 0.00022
```

## Resíduos com distribuição normal

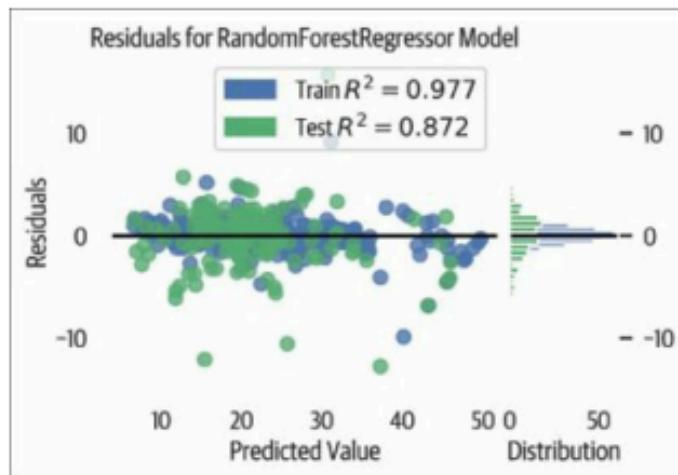
A biblioteca `scipy` inclui um *gráfico de probabilidades* e o teste de *Kolmogorov-Smirnov*; ambos verificam se os resíduos têm distribuição normal.

Podemos gerar um histograma (veja a Figura 15.2) para visualizar os resíduos e verificar se têm distribuição normal:

```
>>> fig, ax = plt.subplots(figsize=(6, 4))
>>> resids = bos_y_test - rfr.predict(bos_X_test)
>>> pd.Series(resids, name="residuals").plot.hist(
... bins=20, ax=ax, title="Residual Histogram"
...
)
>>> fig.savefig("images/mlpr_1502.png", dpi=300)
```

A Figura 15.3 mostra um gráfico de probabilidades. Se as amostras representadas em relação aos quantis estiverem alinhadas, é sinal de que os resíduos têm distribuição normal. Podemos ver que isso não acontece neste caso:

```
>>> from scipy import stats
>>> fig, ax = plt.subplots(figsize=(6, 4))
>>> _ = stats.probplot(resids, plot=ax)
>>> fig.savefig("images/mlpr_1503.png", dpi=300)
```



*Figura 15.1 – Gráfico de resíduos. Outros testes mostrarão que esses dados são heterocedásticos.*

O Yellowbrick é capaz de criar gráficos de resíduos para visualizar esses dados:

```
>>> from yellowbrick.regressor import ResidualsPlot
>>> fig, ax = plt.subplots(figsize=(6, 4))
>>> rpv = ResidualsPlot(rfr)
>>> rpv.fit(bos_X_train, bos_y_train)
>>> rpv.score(bos_X_test, bos_y_test)
>>> rpv.poof()
>>> fig.savefig("images/mlpr_1501.png", dpi=300)
```

## Heterocedasticidade

A biblioteca statsmodel (<https://oreil.ly/HtlI5>) inclui o *teste de Breusch-Pagan* para heterocedasticidade. Isso significa que a variância dos resíduos varia nos valores previstos. No teste de Breusch-Pagan, se os valores p (p-values) forem significativos (p-value menor que 0,05), a hipótese nula da homocedasticidade será rejeitada. Isso mostra que os resíduos são heterocedásticos e que as previsões apresentam distorções.

O teste a seguir confirma a heterocedasticidade:

```
>>> import statsmodels.stats.api as sms
>>> hb = sms.het_breushpagan(resids, bos_X_test)
>>> labels = [
... "Lagrange multiplier statistic",
```

Como no caso do erro médio absoluto, essa medida não é capaz de informar quanto ruim é um modelo, mas pode ser usada para comparar dois modelos. Caso você suponha que os erros estão distribuídos de modo normal, essa será uma boa opção.

O resultado nos informa que, se elevarmos os erros ao quadrado e tirarmos a sua média, o valor resultante estará em torno de 9,5:

```
>>> metrics.mean_squared_error(  
... bos_y_test, bos_y_test_pred  
... )  
9.52886846710526
```

O *erro logarítmico quadrático médio* (na busca em grade, 'neg\_mean\_squared\_log\_error') penaliza a subprevisão, mais do que a superprevisão. Se você tiver alvos que apresentem crescimento exponencial (população, ações etc.), essa é uma boa métrica.

Caso você considere o log do erro e então o eleve ao quadrado, a média desses resultados será igual a 0,021:

```
>>> metrics.mean_squared_log_error(  
... bos_y_test, bos_y_test_pred  
... )  
0.02128263061776433
```

## Gráfico de resíduos

Bons modelos (com pontuações R2 apropriadas) exibirão *homocedasticidade*. Isso significa que a variância é a mesma para todos os valores dos alvos, independentemente da entrada. Ao serem colocados em um gráfico de resíduos, os valores parecerão distribuídos aleatoriamente. Se houver padrões, é sinal de que o modelo ou os dados são problemáticos.

Os gráficos de resíduos também mostram valores discrepantes, que podem ter um grande impacto na adequação do modelo (veja a Figura 15.1).

# Métricas e avaliação de regressão

Neste capítulo, avaliaremos os resultados de uma regressão com floresta aleatória, cujo treinamento foi feito com os dados habitacionais de Boston:

```
>>> rfr = RandomForestRegressor(  
... random_state=42, n_estimators=100  
... )  
>>> rfr.fit(bos_X_train, bos_y_train)
```

## Métricas

O módulo `sklearn.metrics` inclui métricas para avaliar modelos de regressão. As funções de métrica terminadas com `loss` ou `error` devem ser minimizadas. Funções terminadas com `score` devem ser maximizadas.

O *coeficiente de determinação* ( $r^2$ ) é uma métrica de regressão comum. Em geral, esse valor está entre 0 e 1 e representa o percentual da variância do alvo (target) com o qual os atributos contribuem. Valores maiores são melhores, mas, de modo geral, é difícil avaliar o modelo exclusivamente a partir dessa métrica. Um valor igual a 0,7 representa uma boa pontuação? Depende. Para um dado conjunto de dados, 0,5 pode ser uma boa pontuação, enquanto, para outro, um valor igual a 0,9 poderia ser ruim. Geralmente usamos esse número junto com outras métricas ou visualizações para avaliar um modelo.

Por exemplo, é fácil criar um modelo que faça a predição dos preços de ações para o dia seguinte com um  $r^2$  igual a 0,99. No entanto, eu não comprometeria o meu dinheiro com base nesse modelo. Ele poderia fazer previsões um pouco abaixo ou acima, o que significaria um grande problema nas negociações.

A métrica  $r^2$  é a métrica default, usada durante buscas em grade (grid search).

Outras métricas podem ser especificadas com o parâmetro `scoring`.

O método `.score` calcula esse valor para modelos de regressão:

## CAPÍTULO 14

# Regressão

A regressão é um processo de machine learning supervisionado. É semelhante à classificação, mas, em vez de prever um rótulo, tentamos prever um valor contínuo. Se você estiver tentando prever um número, utilize a regressão.

O fato é que o `sklearn` é capaz de aplicar muitos dos mesmos modelos de classificação em problemas de regressão. Com efeito, a API é a mesma e chama `.fit`, `.score` e `.predict`. Isso também vale para as bibliotecas de boosting da próxima geração, como XGBoost e LightGBM.

Embora haja semelhanças quanto aos modelos de classificação e os hiperparâmetros, as métricas de avaliação são diferentes no caso de uma regressão. Neste capítulo, analisaremos vários tipos de modelos de regressão. Usaremos o conjunto de dados habitacionais de Boston (<https://oreil.ly/b2bKQ>) para explorá-los.

A seguir, carregaremos os dados, criaremos uma versão com os dados de treinamento e de teste separados e outra versão diferente com os dados padronizados:

```
>>> import pandas as pd  
>>> from sklearn.datasets import load_boston  
>>> from sklearn import (  
... model_selection,  
... preprocessing,  
... )  
>>> b = load_boston()  
>>> bos_X = pd.DataFrame(  
... b.data, columns=b.feature_names  
... )  
>>> bos_y = b.target  
  
>>> bos_X_train, bos_X_test, bos_y_train, bos_y_test = model_selection.train_test_split(  
... bos_X,
```

# CAPÍTULO 13

## Explicando os modelos

Modelos preditivos têm diferentes propriedades. Alguns foram concebidos para lidar com dados lineares; outros são capazes de modelar dados de entrada mais complexos. Alguns modelos podem ser muito facilmente interpretados, enquanto outros são como caixas-pretas e não oferecem muitos insights acerca de como a predição é feita.

Neste capítulo, veremos como interpretar diferentes modelos. Analisaremos alguns exemplos usando os dados do Titanic.

```
>>> dt = DecisionTreeClassifier(  
... random_state=42, max_depth=3  
... )  
>>> dt.fit(X_train, y_train)
```

### Coeficientes de regressão

Os interceptos e os coeficientes de regressão explicam o valor esperado e como os atributos causam impacto na predição. Um coeficiente positivo indica que, à medida que o valor de um atributo aumenta, o mesmo ocorrerá com a predição.

### Importância dos atributos

Modelos baseados em árvore da biblioteca scikit-learn incluem um atributo `.feature_importances_` para inspecionar como os atributos de um conjunto de dados afetam o modelo. Podemos inspecionar esses dados ou colocá-los em um gráfico.

### LIME

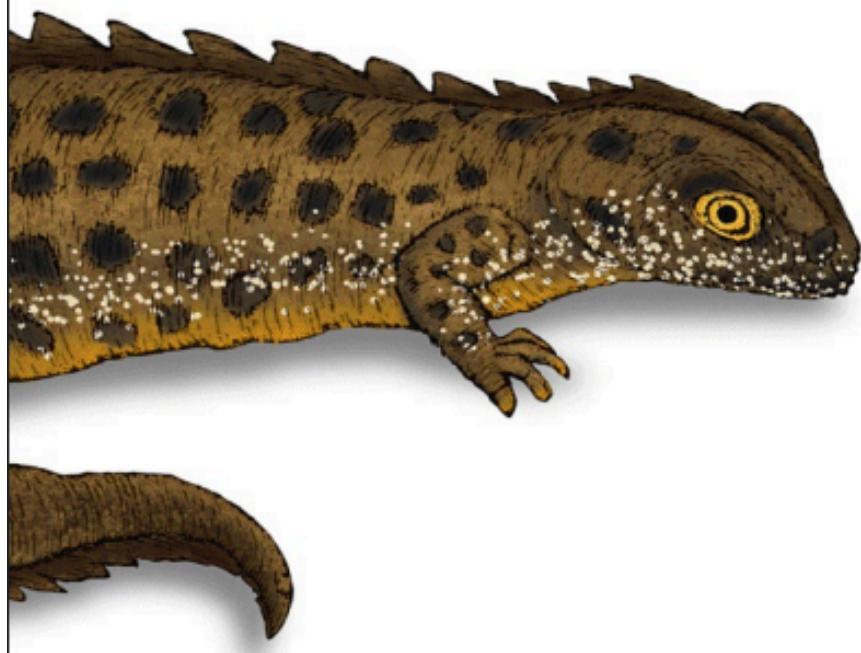
O LIME ([https://oreil.ly/shCR\\_](https://oreil.ly/shCR_)) ajuda a explicar modelos caixa-preta. Ele faz uma interpretação *local*, em vez de uma interpretação geral, e ajuda a explicar

O'REILLY®

# Machine Learning

## Guia de Referência Rápida

Trabalhando com dados estruturados em Python



novatec

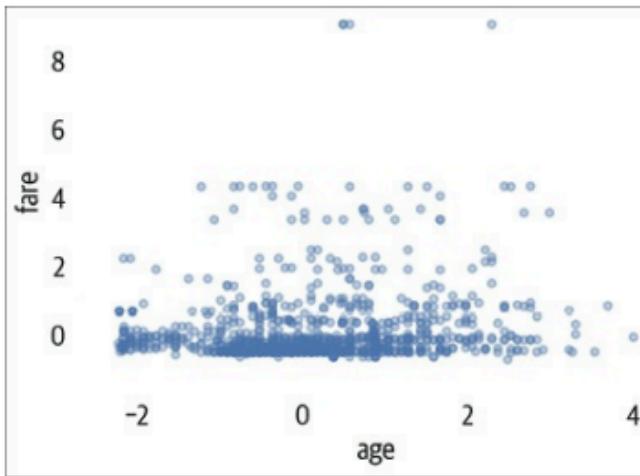
Matt Harrison

*Figura 6.2 – Histograma gerado com o seaborn.*

## Gráfico de dispersão

Um gráfico de dispersão (scatter plot) mostra o relacionamento entre duas colunas numéricas (veja a Figura 6.3). Novamente, isso é fácil de fazer com o pandas. Ajuste o parâmetro alpha caso tenha dados que se sobreponham.

```
>>> fig, ax = plt.subplots(figsize=(6, 4))
>>> X.plot.scatter(
...     x="age", y="fare", ax=ax, alpha=0.3
... )
>>> fig.savefig("images/mlpr_0603.png", dpi=300)
```



*Figura 6.3 – Gráfico de dispersão gerado com o pandas.*

Não parece haver muita correlação entre esses dois atributos. Podemos usar a correlação de Pearson entre duas colunas (pandas) aplicando o método .corr para quantificá-la:

```
>>> X.age.corr(X.fare)
0.17818151568062093
```

## Gráfico conjunto

O Yellowbrick tem um gráfico de dispersão mais sofisticado que inclui histogramas nas bordas e uma linha de regressão, chamada gráfico conjunto (joint plot) – veja a Figura 6.4.

```
>>> from yellowbrick.features import (
```

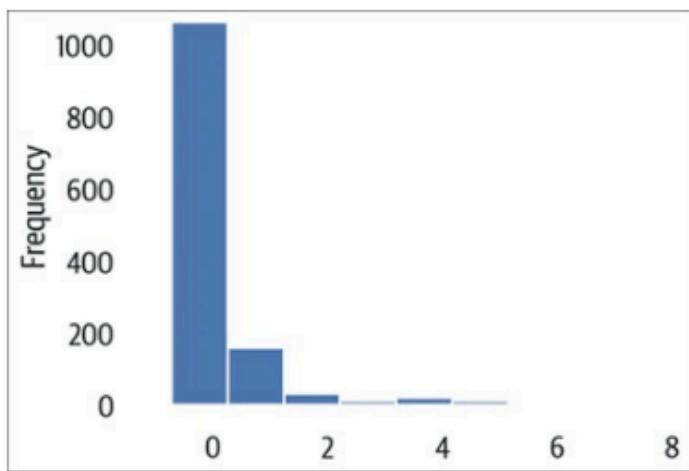
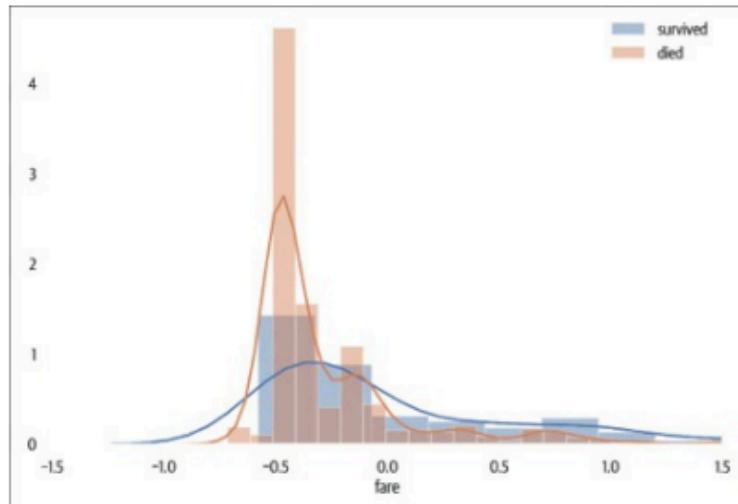


Figura 6.1 – Histograma gerado com o pandas.

Ao usar a biblioteca seaborn, podemos gerar um histograma de valores contínuos em relação ao alvo (veja a Figura 6.2).

```
fig, ax = plt.subplots(figsize=(12, 8))
mask = y_train == 1
ax = sns.distplot(X_train[mask].fare, label='survived')
ax = sns.distplot(X_train[~mask].fare, label='died')
ax.set_xlim(-1.5, 1.5)
ax.legend()
fig.savefig('images/mlpr_0602.png', dpi=300, bbox_inches='tight')
```



ANNs can be seen as just another model in the machine learning life cycle (chapter 8). Figure 9.2 recaps that life cycle. A problem needs to be identified; that data needs to be collected, understood, and prepared; and the ANN model will be tested and improved if necessary.

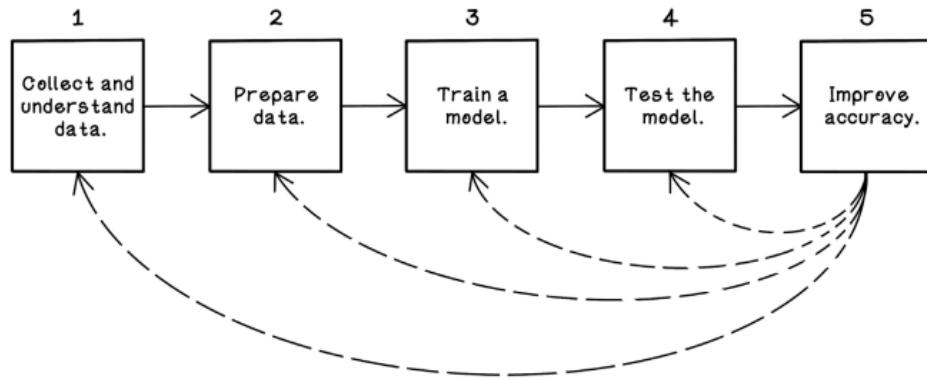


Figure 9.2 A workflow for machine learning experiments and projects

desempenho.

## Curva de validação

Criar uma curva de validação é uma forma de determinar um valor apropriado para um hiperparâmetro. Uma curva de validação é um gráfico que mostra como o desempenho do modelo responde a mudanças no valor do hiperparâmetro (veja a Figura 11.1).

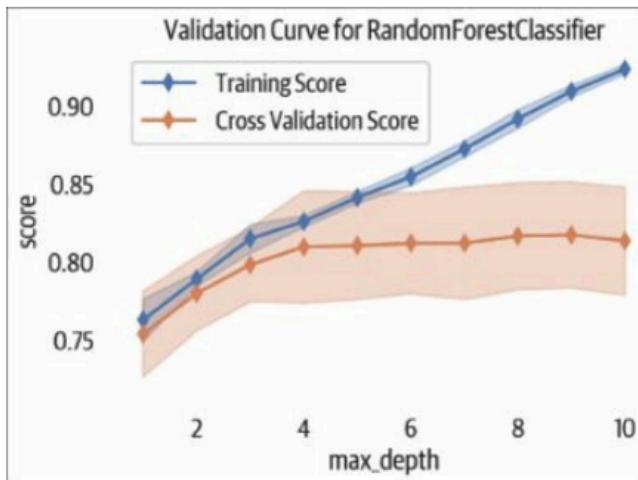


Figura 11.1 – Gráfico da curva de validação.

O gráfico mostra tanto os dados de treinamento como de validação. As pontuações dos dados de validação nos permitem inferir como o modelo responderia a dados não vistos anteriormente. Em geral, escolhemos um hiperparâmetro que maximize a pontuação dos dados de validação.

## Eficiência na execução

A implementação do scikit-learn é  $O(n^4)$ , portanto pode ser difícil escalar para tamanhos maiores. Usar um kernel linear ou o modelo LinearSVC pode melhorar o desempenho da execução, talvez à custa da precisão. Aumentar o valor do parâmetro `cache_size` pode reduzir a ordem para  $O(n^3)$ .

---

default é o Radial Basis Function, ou Função de Base Radial ('rbf'), controlado pelo padrão `gamma`, o qual é capaz de mapear um espaço de entrada em um espaço com mais dimensões.

As SVMs têm as seguintes propriedades:

#### *Eficiência na execução*

A implementação do scikit-learn é  $O(n^4)$ , portanto pode ser difícil escalar para tamanhos maiores. Usar um kernel linear ou o modelo `LinearSVC` pode melhorar o desempenho da execução, talvez à custa da precisão. Aumentar o valor do parâmetro `cache_size` pode reduzir a ordem para  $O(n^3)$ .

#### *Pré-processamento dos dados*

O algoritmo não é invariante à escala. Padronizar os dados é extremamente recomendável.

#### *Para evitar uma superadequação*

O parâmetro `C` (parâmetro de penalidade) controla a regularização. Um valor menor permite ter uma margem menor no hiperplano. Um valor maior para `gamma` tenderá a uma superadequação nos dados de treinamento. O modelo `LinearSVC` aceita parâmetros `loss` e `penalty` para regularização.

#### *Interpretação dos resultados*

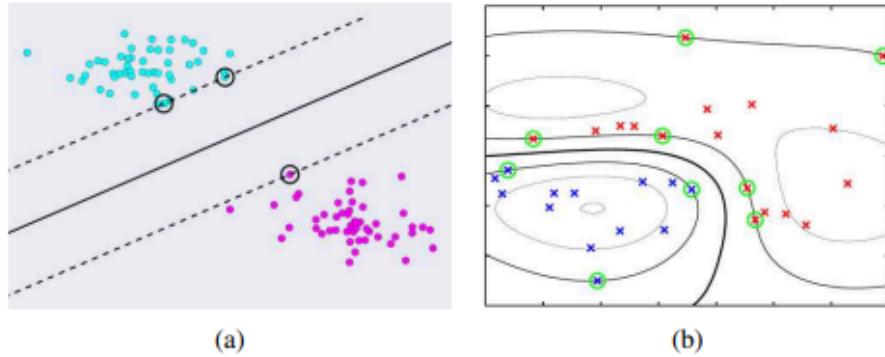
Inspecione `.support_vectors_`, embora possam ser difíceis de explicar. Com kernels lineares, você poderá inspecionar `.coef_`.

Eis um exemplo que usa a implementação de SVM do scikit-learn:

```
>>> from sklearn.svm import SVC  
>>> svc = SVC(random_state=42, probability=True)  
>>> svc.fit(X_train, y_train)  
SVC(C=1.0, cache_size=200, class_weight=None,  
    coef0=0.0, decision_function_shape='ovr',
```

#### 5.1.4 Support vector machines

As we have just mentioned, in some applications of logistic regression we cannot determine a single optimal decision surface (choice of weight and bias vectors  $\{\mathbf{w}_k, b_k\}$  in (5.21)) because there are *gaps* in the feature space where any number of planes could be introduced. Consider Figure 5.11a, where the two classes are denoted in cyan and magenta colors. In addition to the two dashed lines and the solid line, there are infinitely many other lines that will also cleanly separate the two classes, including a swath of horizontal lines. Since the classification error for any of these lines is zero, how can we choose the best decision surface, keeping in mind that we only have a limited number of training examples, and that actual run-time examples



**Figure 5.11** (a) A support vector machine (SVM) finds the linear decision surface (hyperplane) that maximizes the margin to the nearest training examples, which are called the support vectors © Glassner (2018). (b) A two-dimensional two class example of a Gaussian kernel support vector machine (Bishop 2006) © 2006 Springer. The red and blue  $\times$ s indicate the training samples, and the samples circled in green are the support vectors. The black lines indicate iso-contours of the kernel regression function, with the contours containing the blue and red support vectors indicating the  $\pm 1$  contours and the dark contour in between being the decision surface.

### 5.1.3 Logistic regression

In the previous section, we derived classification rules based on posterior probabilities applied to multivariate Gaussian distributions. Quite often, however, Gaussians are not appropriate models of our class distributions and we must resort to alternative techniques.

One of the simplest among these is *logistic regression*, which applies the same ideas as in the previous section, i.e., a linear projection onto a weight vector,

$$l_i = \mathbf{w} \cdot \mathbf{x}_i + b \quad (5.17)$$

followed by a logistic function

$$p_i = p(\mathcal{C}_0|\mathbf{x}_i) = \sigma(l_i) = \sigma(\mathbf{w}^T \mathbf{x}_i + b) \quad (5.18)$$

to obtain (binary) class probabilities. Logistic regression is a simple example of a *discriminative model*, since it does not construct or assume a prior distribution over unknowns, and hence is not *generative*, i.e., we cannot generate random samples from the class (Bishop 2006, Section 1.5.4).

As we no longer have analytic estimates for the class means and covariances (or they are poor models of the class distributions), we need some other method to determine the weights  $\mathbf{w}$  and bias  $b$ . We do this by maximizing the posterior log likelihoods of the correct labels.

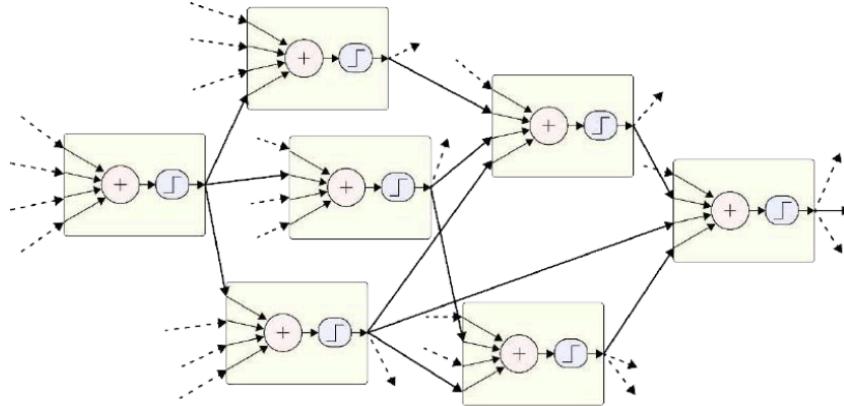
For the binary classification task, let  $t_i \in \{0, 1\}$  be the class label for each training sample  $\mathbf{x}_i$  and  $p_i = p(\mathcal{C}_0|\mathbf{x})$  be the estimated likelihood predicted by (5.18) for a given

weight and bias  $(\mathbf{w}, b)$ . We can maximize the likelihood of the correct labels being predicted by minimizing the negative log likelihood, i.e., the *cross-entropy loss* or error function,

$$E_{\text{CE}}(\mathbf{w}, b) = - \sum_i \{t_i \log p_i + (1 - t_i) \log(1 - p_i)\} \quad (5.19)$$

(Bishop 2006, Section 4.3.2).<sup>9</sup> Note how whenever the label  $t_i = 0$ , we want  $p_i = p(\mathcal{C}_0|\mathbf{x}_i)$  to be high, and vice versa.

This formula can easily be extended to a multi-class loss by again defining the posterior probabilities as normalized exponentials over per-class linear regressions, as in (5.3) and (5.13).



**Figure 5.24** A multi-layer network, showing how the outputs of one unit are fed into additional units. © Glassner (2018)

### 5.1.3 Logistic regression

In the previous section, we derived classification rules based on posterior probabilities applied to multivariate Gaussian distributions. Quite often, however, Gaussians are not appropriate models of our class distributions and we must resort to alternative techniques.

One of the simplest among these is *logistic regression*, which applies the same ideas as in the previous section, i.e., a linear projection onto a weight vector,

$$l_i = \mathbf{w} \cdot \mathbf{x}_i + b \quad (5.17)$$

followed by a logistic function

$$p_i = p(C_0|\mathbf{x}_i) = \sigma(l_i) = \sigma(\mathbf{w}^T \mathbf{x}_i + b) \quad (5.18)$$

to obtain (binary) class probabilities. Logistic regression is a simple example of a *discriminative model*, since it does not construct or assume a prior distribution over unknowns, and hence is not *generative*, i.e., we cannot generate random samples from the class (Bishop 2006, Section 1.5.4).

As we no longer have analytic estimates for the class means and covariances (or they are poor models of the class distributions), we need some other method to determine the weights  $\mathbf{w}$  and bias  $b$ . We do this by maximizing the posterior log likelihoods of the correct labels.

For the binary classification task, let  $t_i \in \{0, 1\}$  be the class label for each training sample  $\mathbf{x}_i$  and  $p_i = p(C_0|\mathbf{x})$  be the estimated likelihood predicted by (5.18) for a given

### 5.1.2 Bayesian classification

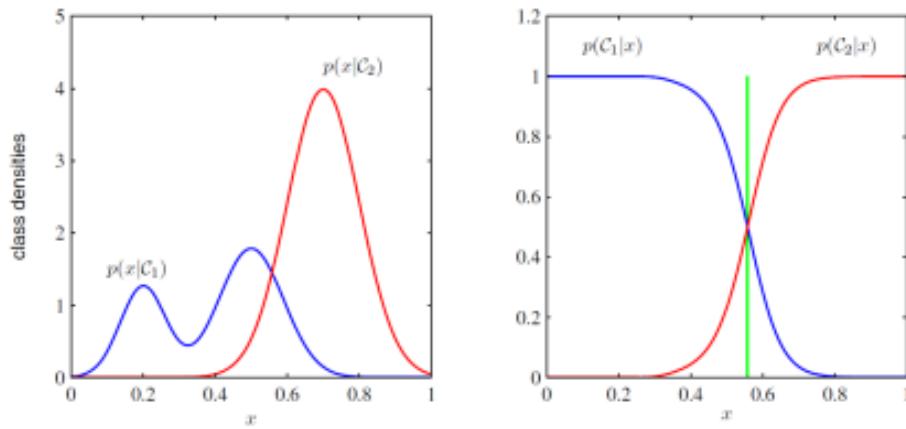
For some simple machine learning problems, e.g., if we have an analytic model of feature construction and noising, or if we can gather enough samples, we can determine the probability distributions of the feature vectors for each class  $p(\mathbf{x}|\mathcal{C}_k)$  as well as the prior class likelihoods  $p(\mathcal{C}_k)$ .<sup>5</sup> According to Bayes' rule (4.33), the likelihood of class  $\mathcal{C}_k$  given a feature vector  $\mathbf{x}$  (Figure 5.6) is given by

$$p_k = p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \quad (5.2)$$

$$= \frac{\exp l_k}{\sum_j \exp l_j}, \quad (5.3)$$

<sup>4</sup><https://github.com/facebookresearch/faiss>

<sup>5</sup>The following notation and equations are adapted from Bishop (2006, Section 4.2), which describes *probabilistic generative classification*.



**Figure 5.6** An example with two class conditional densities  $p(x|\mathcal{C}_k)$  along with the corresponding posterior class probabilities  $p(\mathcal{C}_k|x)$ , which can be obtained using Bayes' rule, i.e., by dividing by the sum of the two curves (Bishop 2006) © 2006 Springer. The vertical green line is the optimal decision boundary for minimizing the misclassification rate.

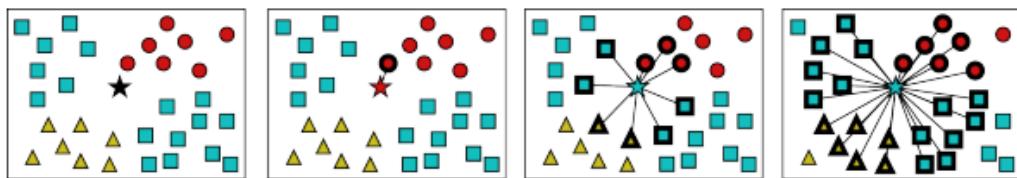
### 5.1.1 Nearest neighbors

Nearest neighbors is a very simple *non-parametric* technique, i.e., one that does not involve a low-parameter analytic form for the underlying distribution. Instead, the training examples are all retained, and at evaluation time the “nearest”  $k$  neighbors are found and then averaged to produce the output.<sup>3</sup>

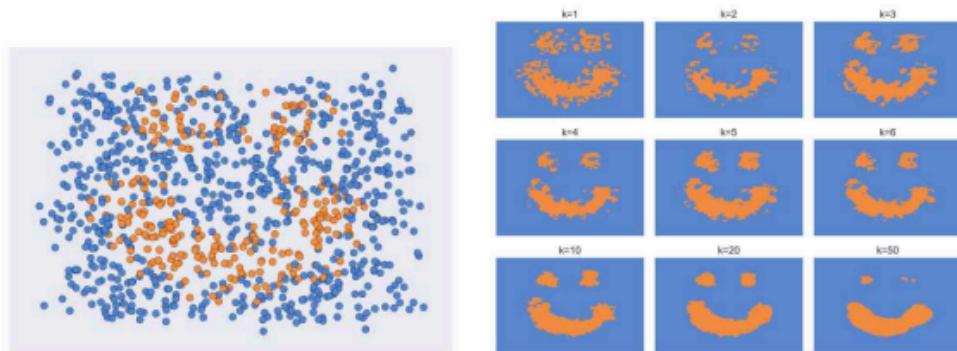
Figure 5.4 shows a simple graphical example for various values of  $k$ , i.e., from using the  $k = 1$  nearest neighbor all the way to finding the  $k = 25$  nearest neighbors and selecting

---

<sup>3</sup>The reason I put “nearest” in quotations is that standardizing and/or whitening the data will affect distances between vectors, and is usually helpful.



**Figure 5.4** Nearest neighbor classification. To determine the class of the star ( $\star$ ) test sample, we find the  $k$  nearest neighbors and select the most popular class. This figure shows the results for  $k = 1, 9$ , and  $25$  samples. © Glassner (2018)



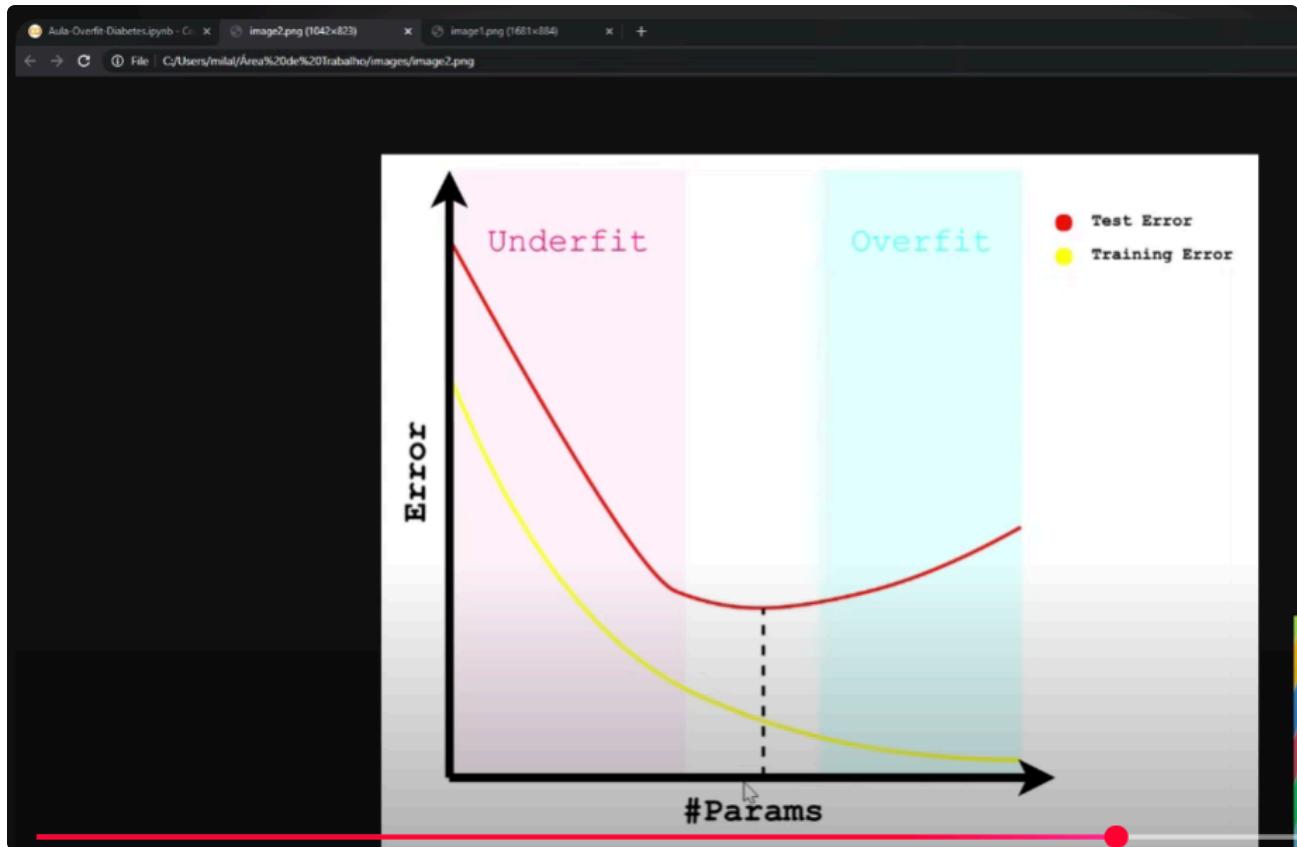
Underfitting

Overfitting

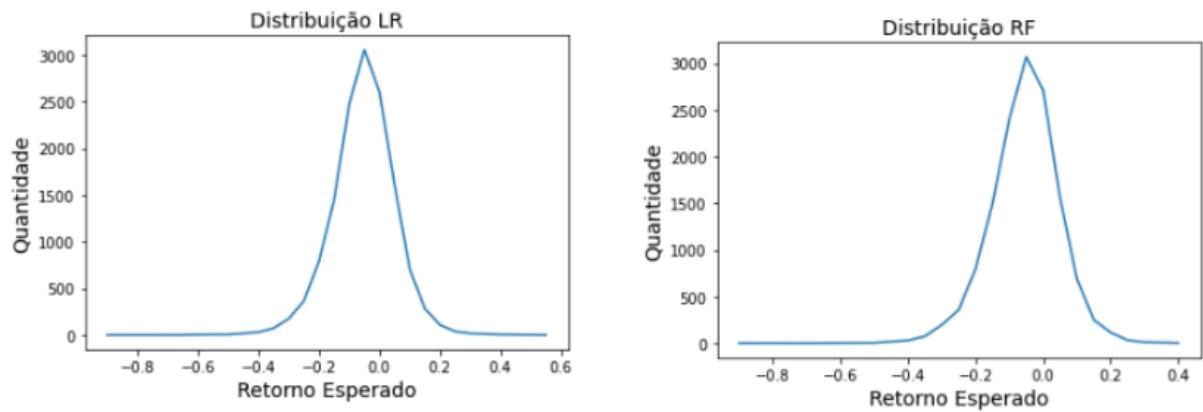


Poucos parâmetros

Muitos parâmetros



**Figura 4.8: Distribuição das previsões em razão do resultado esperado**



39

**Figura 4.7: Avaliação Geral dos Modelos**

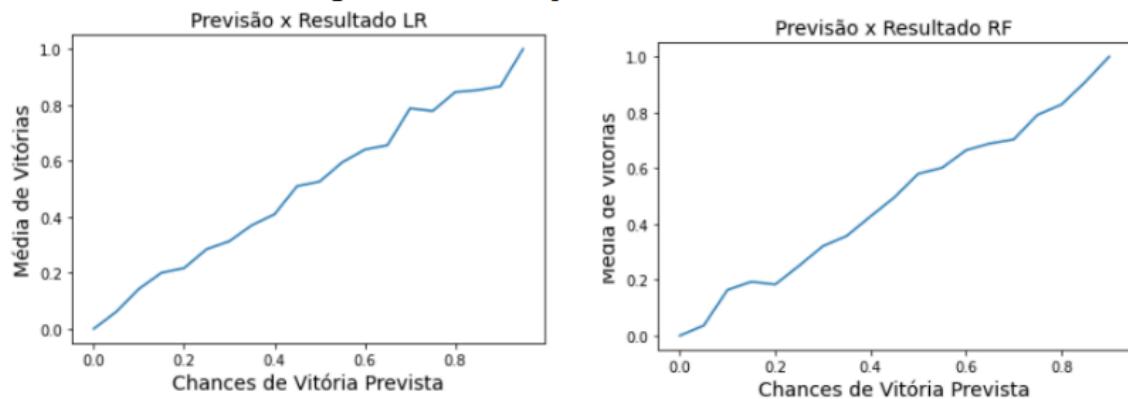


Figura 4.3: Conjunto de parâmetros utilizados RF

```
max_depth = [1, 10, 50, 100, 200, 300, 400]
min_samples_split = [1, 2, 5, 10, 15, 20, 30]
min_samples_leaf = [1, 2, 3, 4, 8, 12]
bootstrap = [True, False]
criterions=['gini', 'entropy']
random_grid = {'n_estimators': n_estimators,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap,
               'criterion': criterion}
```

Figura 4.4: Conjunto de parâmetros utilizados LR

```
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
C_values = [100, 10, 1.0, 0.1, 0.01]
grid = dict(solvers=solvers, C=C_values)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)
grid_result = grid_search.fit(X_train[features], y_train)
```

Figura 4.5: Hiperparâmetros selecionados RF

```
model = RandomForestClassifier(n_estimators= 300,
                             min_samples_split= 10,
                             min_samples_leaf= 3,
                             max_depth= 12,
                             criterions= 'entropy',
                             bootstrap= True)
```

Figura 4.6: Hiperparâmetros selecionados LR

```
model = LogisticRegression(C= 100, solver= 'newton-cg')
```

Figura 4.2: Validação cruzada - Média e Desvio

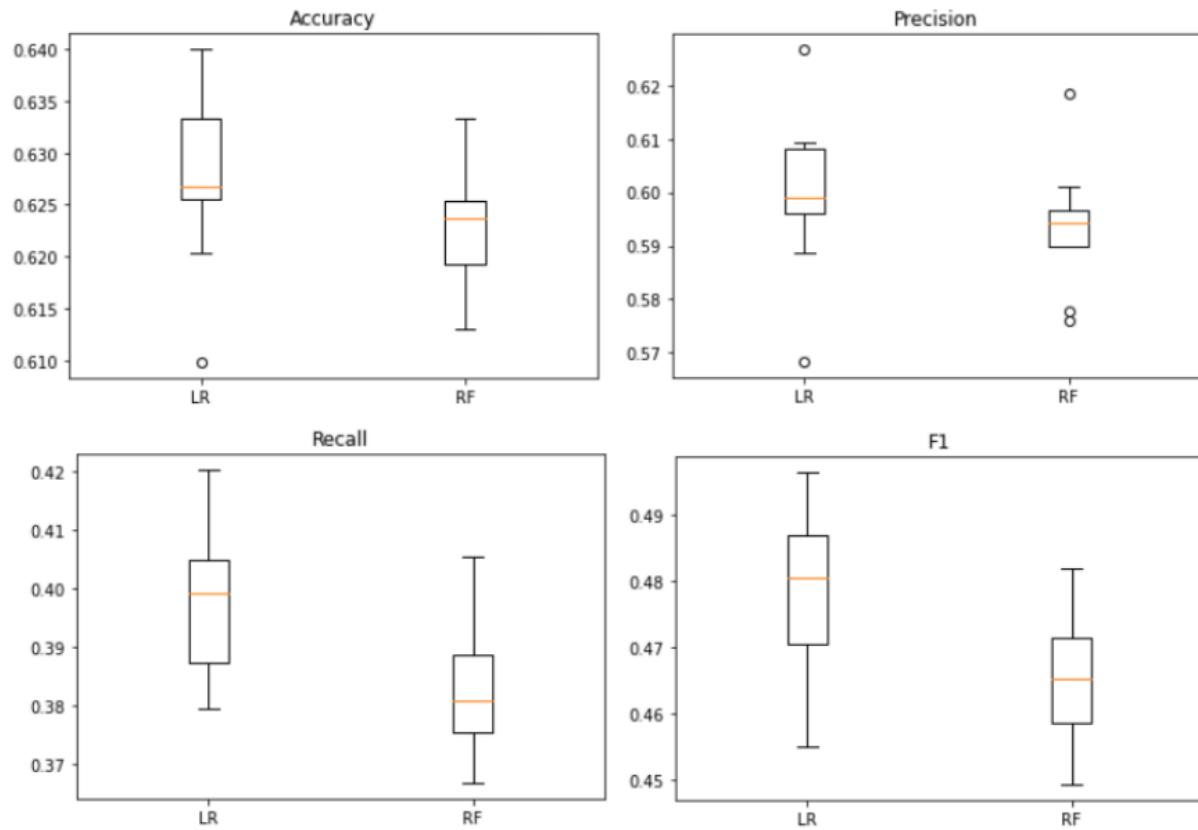


Tabela 4.8: Metricas de Validação

Modelo	Accuracy	Precision	Recall	f1	ROC
Logistic Regression	0.627	0.600	0.398	0.478	0.658
K Neighbors	0.554	0.477	0.410	0.441	0.551
Decision Tree	0.548	0.474	0.480	0.477	0.540
Gaussian NB	0.608	0.541	0.581	0.560	0.650
XGB	0.610	0.561	0.423	0.482	0.633
Random Forest	0.622	0.592	0.393	0.466	0.648
Neural Network	0.624	0.590	0.410	0.492	0.600

Figura 3.4: Cotações x Resultado

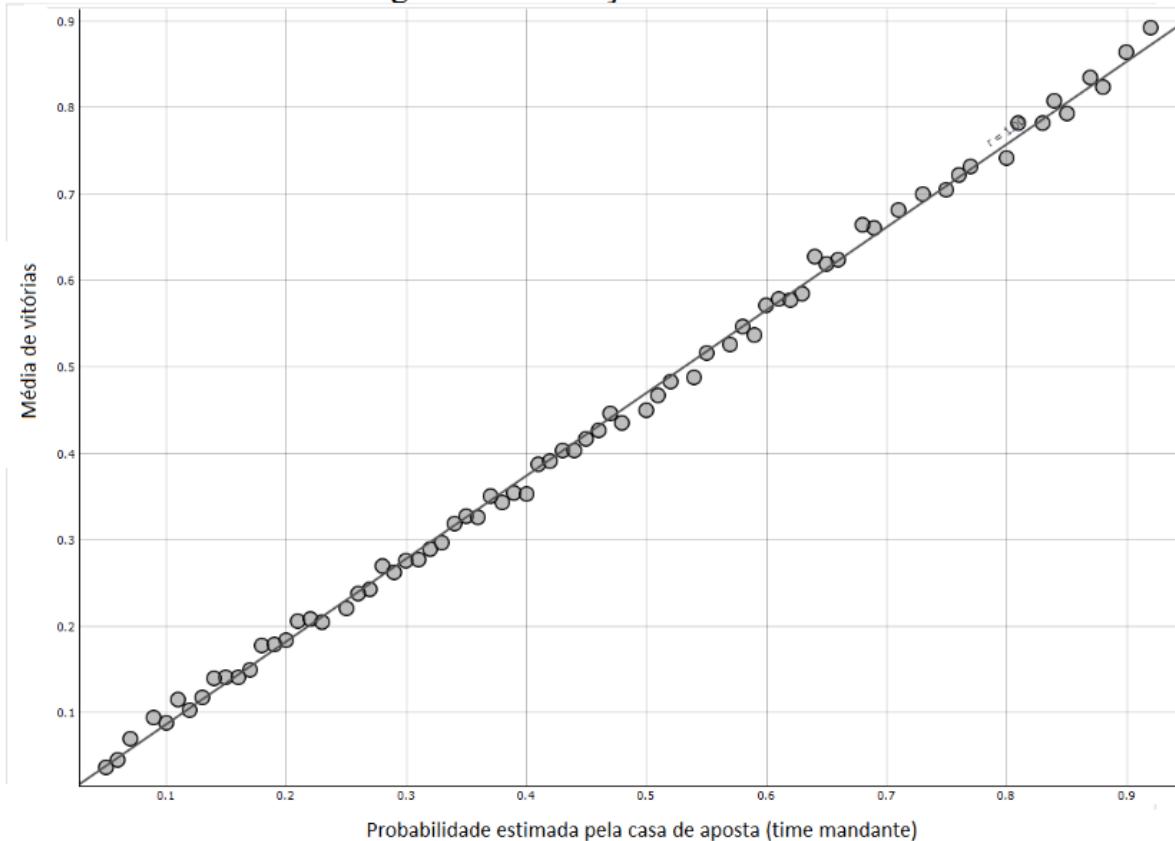


Figura 3.3: Diferença de Gols

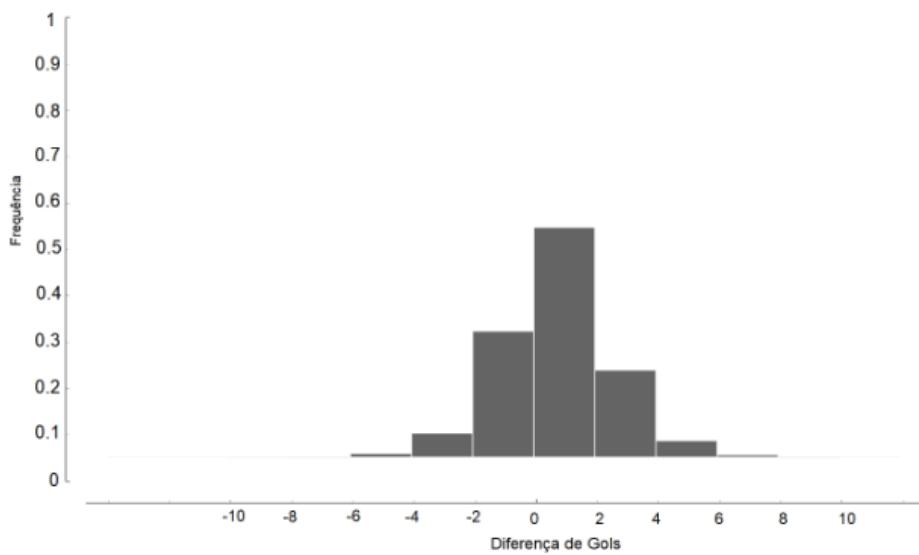
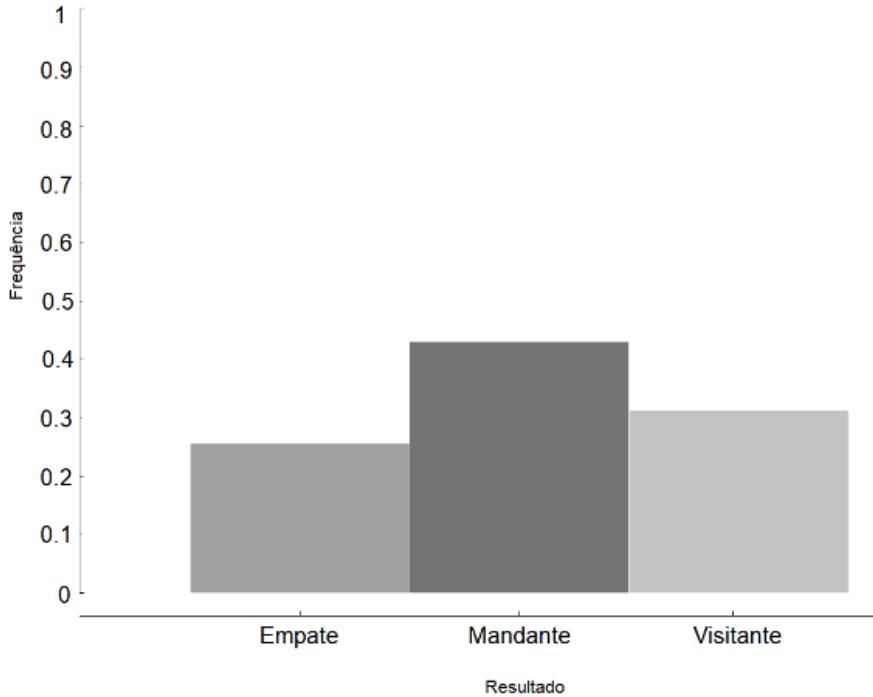
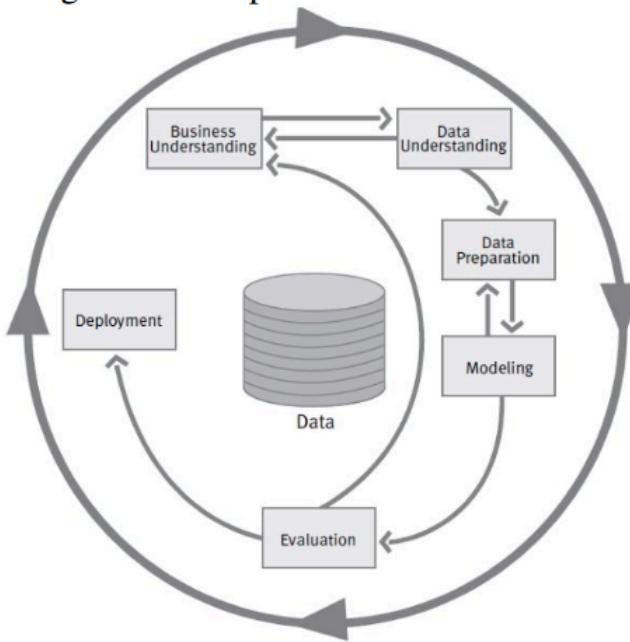


Figura 3.2: Distribuição dos resultados em razão do vencedor do confronto

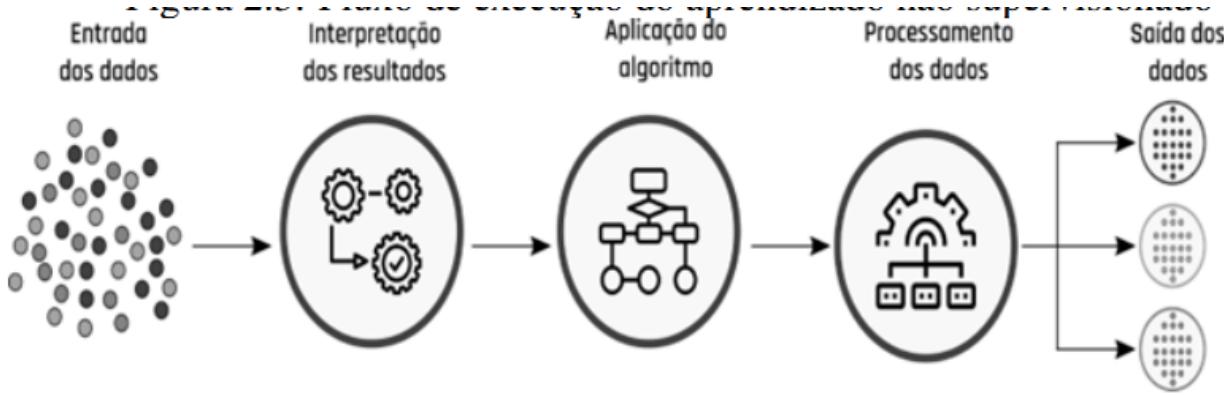


A metodologia *CRISP DM* define o ciclo de vida do projeto, dividindo-o nas seguintes etapas: Entendimento do problema, Compreensão dos dados, Preparação dos dados, Modelagem e Avaliação.

Figura 3.1: Etapas da modelo CRISP DM

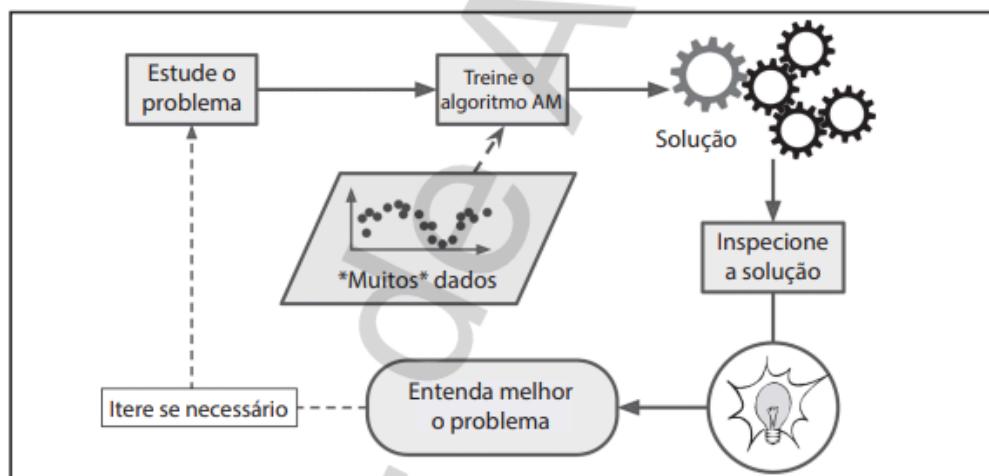


Fonte: Jounal of Data Warehousing (2000)



Finalmente, o Aprendizado de Máquina pode ajudar os seres humanos a aprender (Figura 1-4): os algoritmos do AM podem ser inspecionados para que vejamos o que eles aprenderam (embora para alguns algoritmos isso possa ser complicado). Por exemplo, uma vez que o filtro foi treinado para o spam, ele pode facilmente ser inspecionado e revelar uma lista de palavras e combinações previstas que ele acredita serem as mais prováveis. Às vezes, isso revelará correlações não esperadas ou novas tendências, levando a uma melhor compreensão do problema.

Aplicar técnicas do AM para se aprofundar em grandes quantidades de dados pode ajudar na descoberta de padrões que não eram aparentes. Isto é chamado de mineração de dados.



*Figura 1-4. O Aprendizado de Máquina pode ajudar no ensino dos humanos*

Resumindo, o Aprendizado de Máquina é ótimo para:

- Problemas para os quais as soluções existentes exigem muita configuração manual ou longas listas de regras: um algoritmo de Aprendizado de Máquina geralmente simplifica e melhora o código;
- Problemas complexos para os quais não existe uma boa solução quando utilizamos uma abordagem tradicional: as melhores técnicas de Aprendizado de Máquina podem encontrar uma solução;

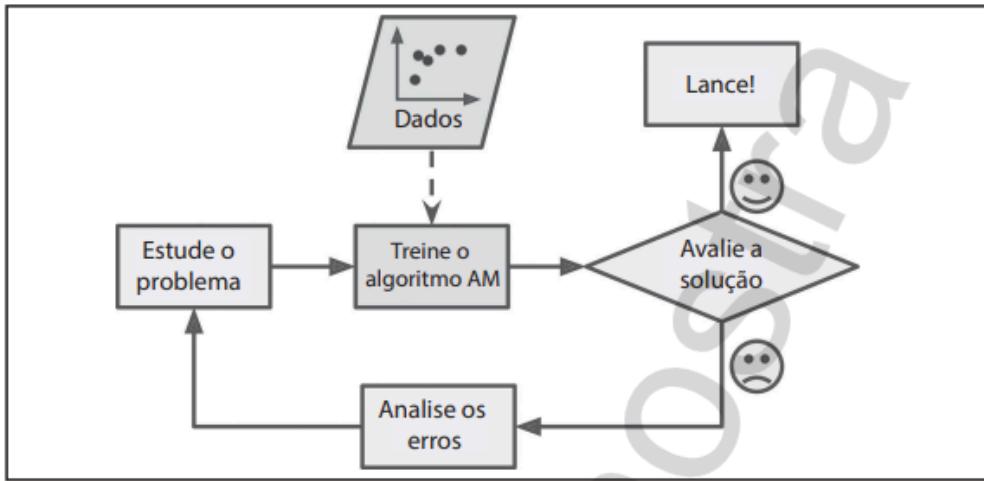


Figura 1-2. Abordagem do Aprendizado de Máquina

Além disso, se os *spammers* perceberem que todos os seus e-mails contendo “4U” são bloqueados, poderão começar a escrever “For U”. Um filtro de *spam* que utiliza técnicas de programação tradicionais precisaria ser atualizado para marcar os e-mails “For U”. Se os *spammers* continuam contornando seu filtro de *spam*, será preciso escrever novas regras para sempre.

Em contrapartida, um filtro de *spam* baseado em técnicas de Aprendizado de Máquina percebe automaticamente que “For U” tornou-se frequente no *spam* marcado pelos usuários e começa a marcá-los sem a sua intervenção (Figura 1-3).

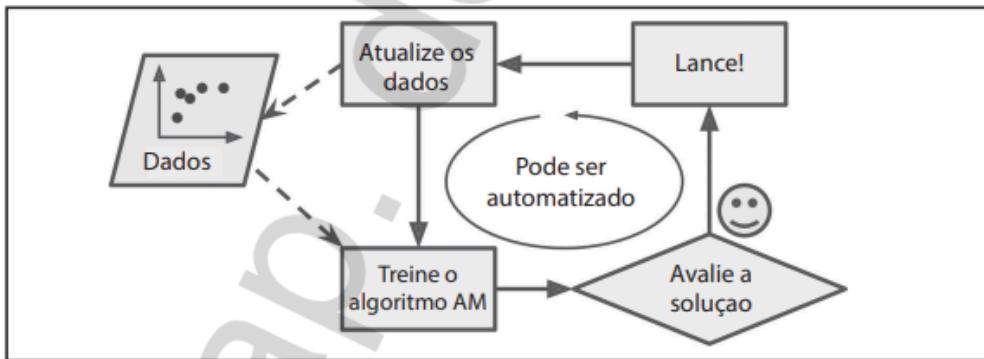


Figura 1-3. Adaptando-se automaticamente à mudança

Outra área na qual o Aprendizado de Máquina se destaca é nos problemas muito com-

Página inicial

Crash Course

dados.

Filtrar

Perda (10 min)

Exercício interativo: parâmetros (5 min)

Gradiente descendente (10 min)

Hiperparâmetros (10 min)

Exercício interativo: gradiente descendente (5 min)

&lt;&gt; Exercício de programação (10 min)

Teste seus conhecimentos (10 min)

➡ A seguir

▶ Regressão logística (35 min)

▶ Classificação (70 min)

Dados

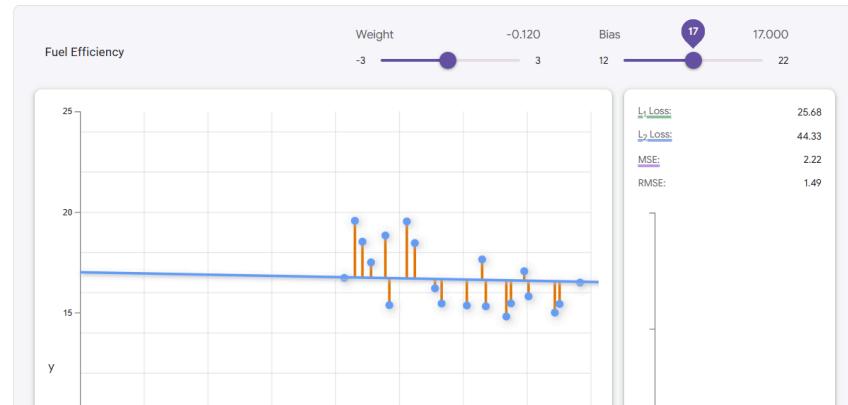
▶ Trabalhar com dados numéricos (85 min)

▶ Como trabalhar com dados categóricos (50 min)

▶ conjuntos de dados, generalização e overfitting (105 min)

## Perguntas a serem consideradas:

- Qual é o MSE mais baixo que você pode alcançar?
- Quais valores de peso e viés produziram essa perda?



Filtrar

Perda (10 min)

Exercício interativo: parâmetros (5 min)

Gradiente descendente (10 min)

Hiperparâmetros (10 min)

Exercício interativo: gradiente descendente (5 min)

&lt;&gt; Exercício de programação (10 min)

Teste seus conhecimentos (10 min)

➡ A seguir

▶ Regressão logística (35 min)

▶ Classificação (70 min)

Dados

▶ Trabalhar com dados numéricos (85 min)

▶ Como trabalhar com dados categóricos (50 min)

▶ conjuntos de dados, generalização e overfitting (105 min)

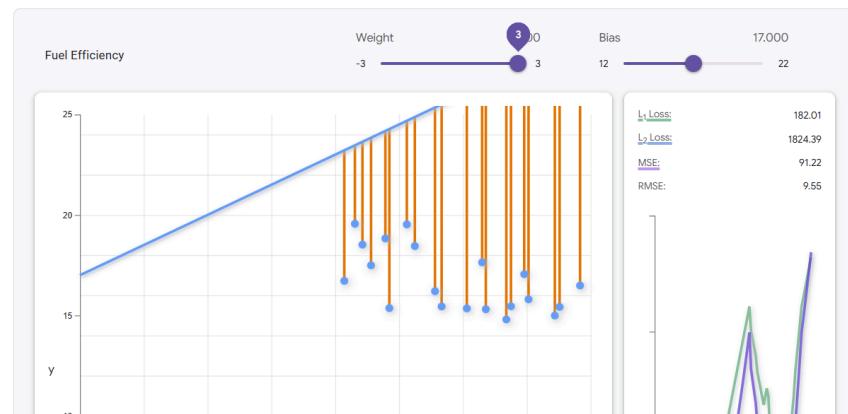
Modelos de ML avançado

▶ Redes neurais (75 min)

Sua tarefa: ajuste os controles deslizantes Peso e Viés acima do gráfico para encontrar o modelo linear que minimiza a perda de MSE nos dados.

## Perguntas a serem consideradas:

- Qual é o MSE mais baixo que você pode alcançar?
- Quais valores de peso e viés produziram essa perda?



Filtrar

- Perda (10 min)
- Exercício interativo: parâmetros (5 min)**
- Gradiente descendente (10 min)
- Hiperparâmetros (10 min)
- Exercício interativo: gradiente descendente (5 min)
- ↔ Exercício de programação (10 min)
- Teste seus conhecimentos (10 min)
- ➡ A seguir
- ▶ Regressão logística (35 min)
- ▶ Classificação (70 min)

## Dados

- ▶ Trabalhar com dados numéricos (85 min)
- ▶ Como trabalhar com dados categóricos (50 min)
- ▶ conjuntos de dados, generalização e overfitting (105 min)

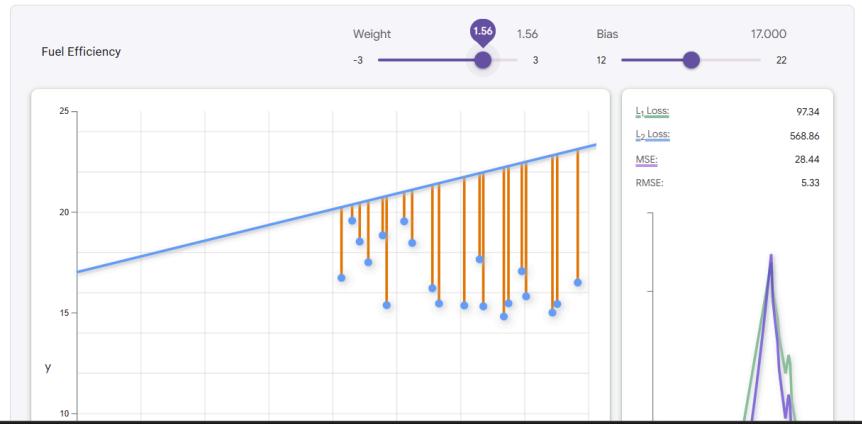
## Modelos de ML avançado

- ▶ Redes neurais (75 min)

**Sua tarefa:** ajuste os controles deslizantes Peso e Viés acima do gráfico para encontrar o modelo linear que minimiza a perda de MSE nos dados.

## Perguntas a serem consideradas:

- Qual é o MSE mais baixo que você pode alcançar?
- Quais valores de peso e viés produziram essa perda?



Página inicial

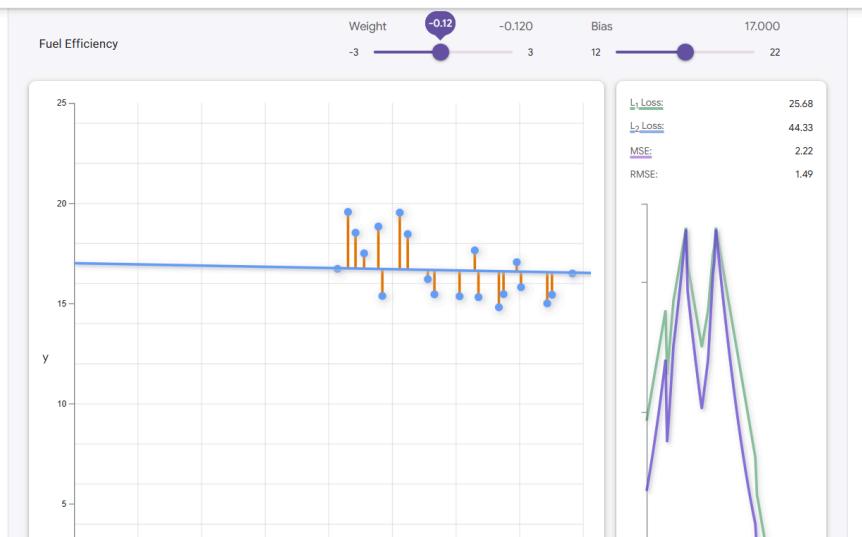
Crash course

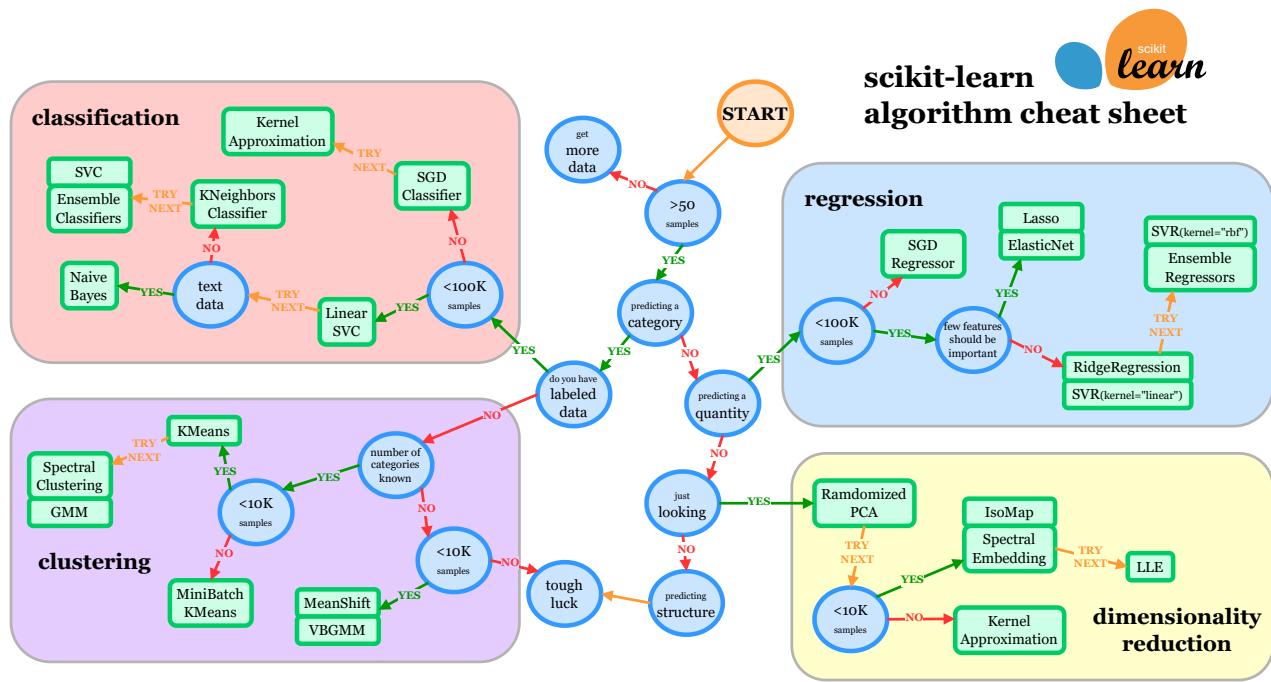
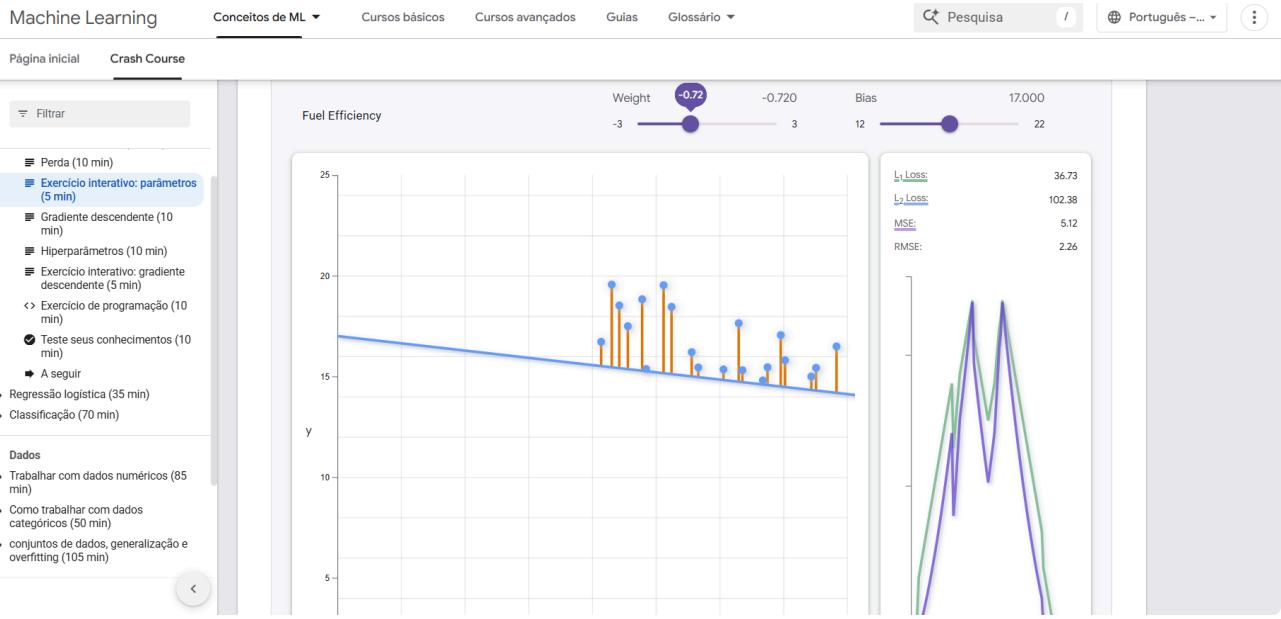
Filtrar

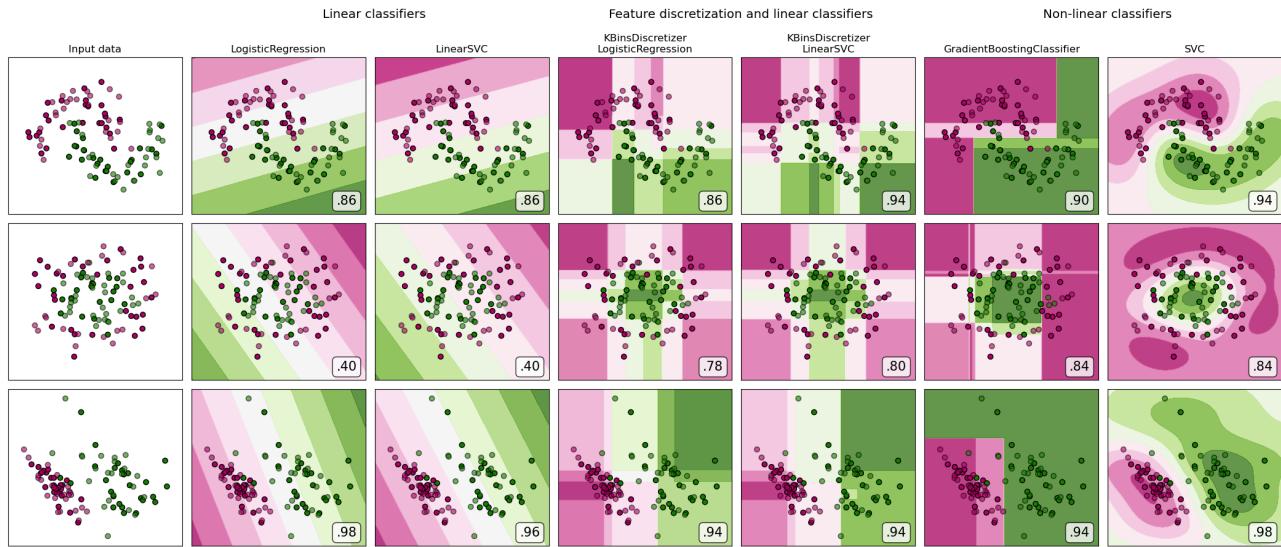
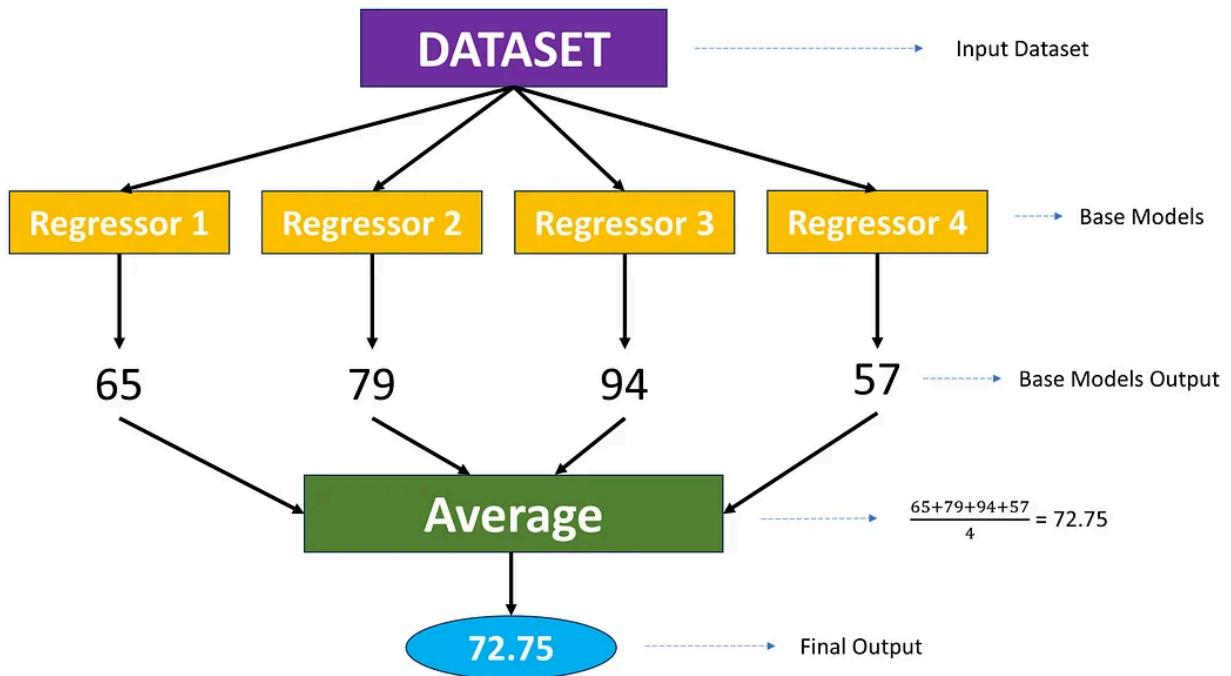
- Perda (10 min)
- Exercício interativo: parâmetros (5 min)**
- Gradiente descendente (10 min)
- Hiperparâmetros (10 min)
- Exercício interativo: gradiente descendente (5 min)
- ↔ Exercício de programação (10 min)
- Teste seus conhecimentos (10 min)
- ➡ A seguir
- ▶ Regressão logística (35 min)
- ▶ Classificação (70 min)

## Dados

- ▶ Trabalhar com dados numéricos (85 min)
- ▶ Como trabalhar com dados categóricos (50 min)
- ▶ conjuntos de dados, generalização e overfitting (105 min)







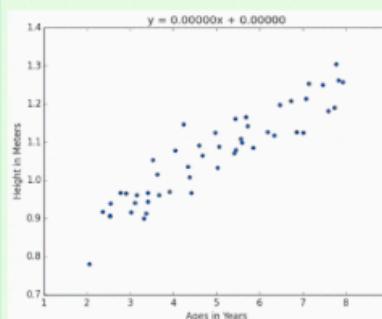


# Top Machine Learning Algorithms

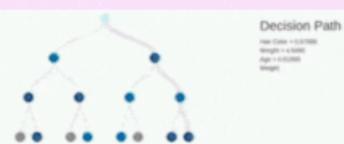
## K Means Clustering



## Linear Regression



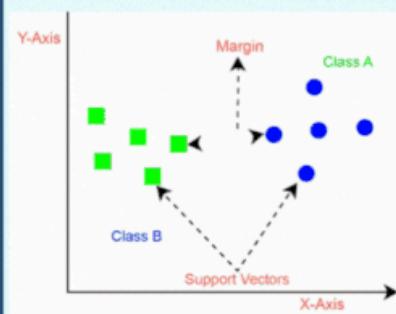
## Decision Tree



## Logistic Regression



## SVM



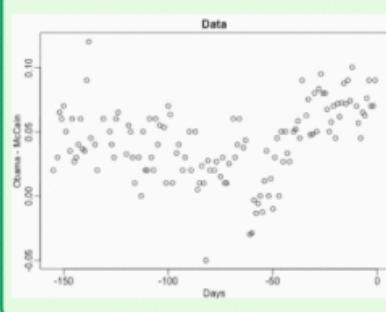
## Naive Bayes



## KNN



## Random Forest



## Dimensionality Reduction Algorithms

