Modelagem Analítica com Machine Learning

Quinto Dia

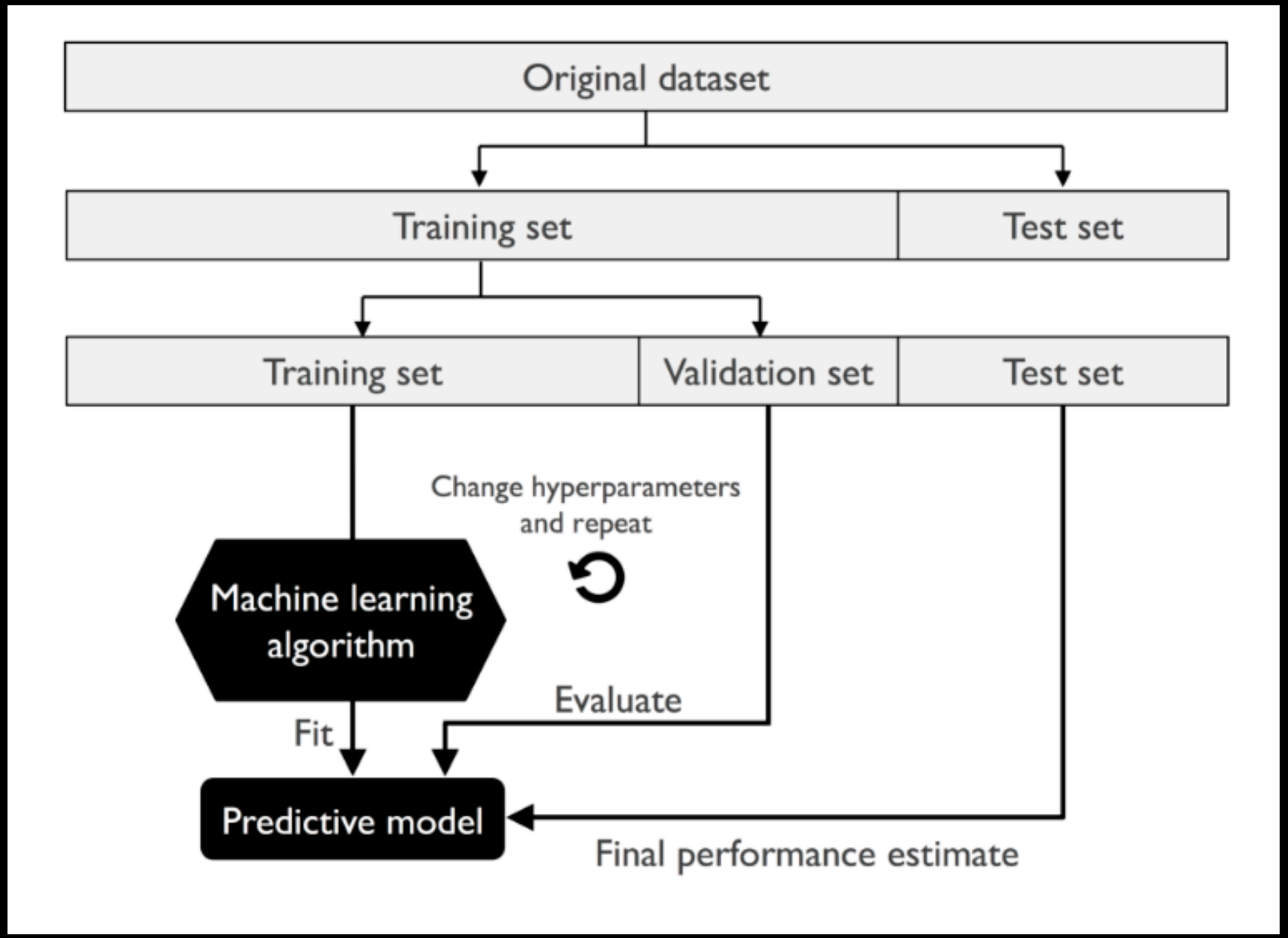
Paulo Cysne Rios, Jr.

Exercício de Ontem

- Use SVM na predição de qualidade no conjunto de dados Red Wine.
- Use valores diferentes do pârametro C e veja o impacto.
- Divida o conjunto em Treinamento (80%) e Teste (20%).
- Use Validação Cruzada.
- Faça uma Tabela de Confusão, Use Acurácia, Precisão e Recall.
- Compare os resultados com aqueles de Decision Tree e Random Forests.

Avaliação de Modelos

Avaliações do Modelo



Curvas de Aprendizagem

- Se um modelo funciona bem nos dados de treinamento, mas generaliza mal de acordo com as métricas de validação cruzada, seu modelo está overfitting. Se isso ocorre mal em ambos, então ele está underfitting. Esta é uma maneira de saber quando um modelo ou é muito simples ou muito complexo.

- Outra maneira é olhar a sua curva de aprendizagem.

Curvas de Aprendizagem

- São gráficos do desempenho do modelo no conjunto de treinamento e no conjunto de validação em função do tamanho do conjunto de treinamento.

- Para gerar estes gráficos, simplesmente treine o modelo várias vezes em subconjuntos de diferentes tamanhos do conjunto de treinamento.

Curvas de Aprendizagem

Exemplo

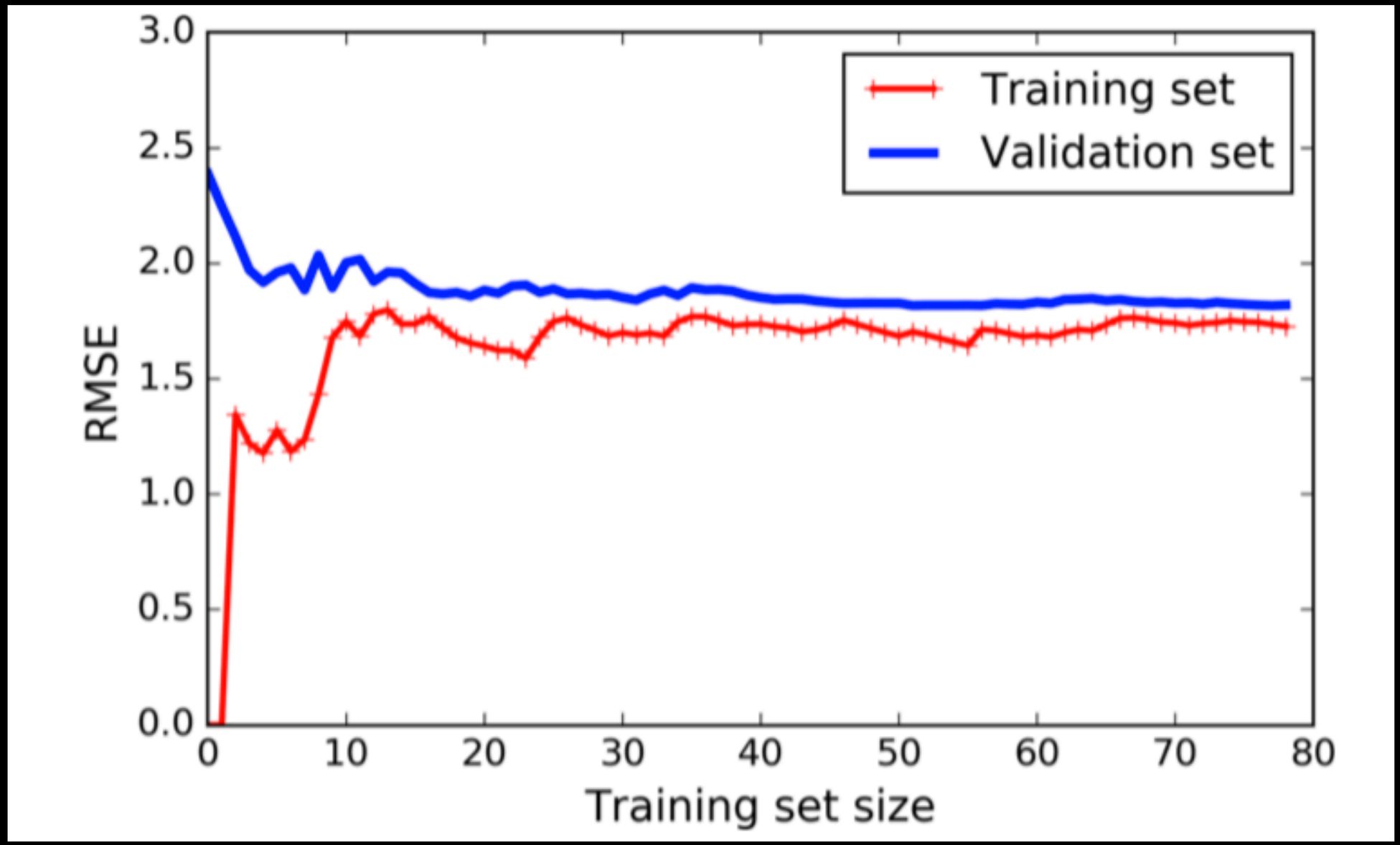
```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

Curvas de Aprendizagem

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
```

```
lin_reg = LinearRegression()
plot_learning_curves(lin_reg, X, y)
```

Curvas de Aprendizagem



Curvas de Aprendizagem

Se seu modelo estiver underfitting
nos dados de treinamento,
adicionar mais exemplos de treinamento
não ajudará.

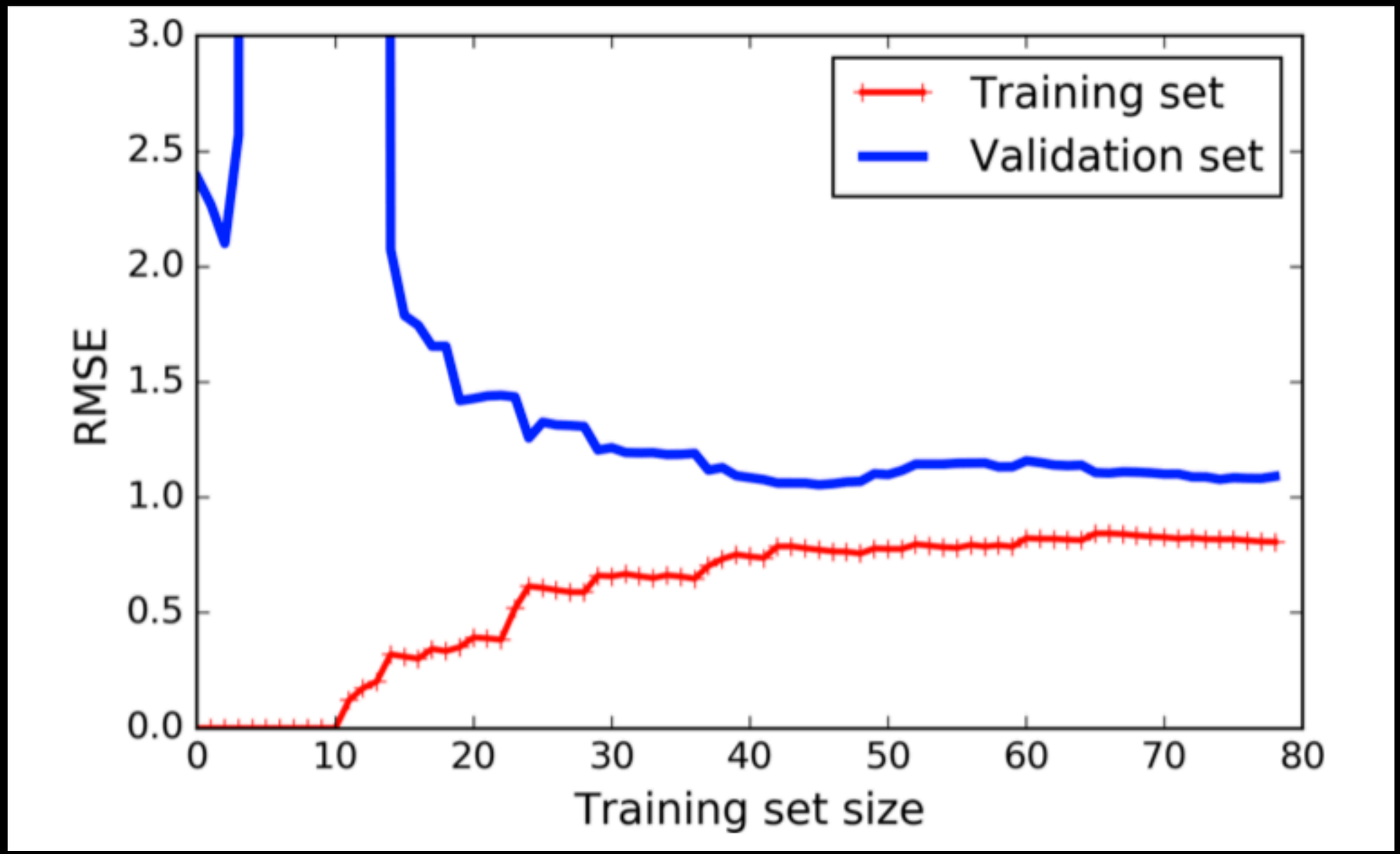
Você precisa usar
um modelo mais complexo
ou ter melhores
variáveis independentes.

Curvas de Aprendizagem

Exemplo

```
from sklearn.pipeline import Pipeline

polynomial_regression = Pipeline([
        ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
        ("lin_reg", LinearRegression()),
    ])

plot_learning_curves(polynomial_regression, X, y)
```

Curvas de Aprendizagem



Curvas de Aprendizagem

Uma maneira de melhorar
um modelo de overfitting
é alimentá-lo com
mais dados de treinamento
até que o erro de validação
atinja o erro de treinamento.

Regularização de Regressão Linear

Regularização de Modelos

- Uma boa maneira de reduzir o overfitting é regularizar o modelo (isto é, para restringi-lo): quanto menor for o grau de liberdade, mais difícil será para superar os dados. Por exemplo, uma maneira simples de regularizar um modelo polinomial é reduzir o número de graus polinomiais.

- Para um modelo linear, a regularização é normalmente alcançada ao restringir os pesos do modelo. Vamos agora ver Ridge Regression, Lasso Regression e Elastic Net, que implementam três maneiras diferentes de restringir os pesos.

Regularização de Modelos de Regressão Linear

- Um termo é adicionado a função de custo. Isso força o algoritmo de aprendizagem a não apenas ter que ajustar os dados, mas também a manter os pesos do modelo tão pequenos quanto possível.

- Observe que o termo de regularização só deve ser adicionado à função de custo durante o treinamento. Uma vez que o modelo é treinado, você deseja avaliar o desempenho do modelo usando a medida de desempenho não regularizada.

Ridge Regression

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Função de custo

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A} \right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

Solução para Minimizar a função custo

- É importante dimensionar os dados (por exemplo, usando um StandardScaler) antes de executar Ridge Regression, pois é sensível à escala das variáveis. Isso é assim para a maioria dos modelos regularizados.

Ridge Regression

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[ 1.55071465]])
```

Com a Equação

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([ 1.13500145])
```

Como a Descida de Gradiente Estocástico (SGD)

Lasso Regression

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Função de custo

- Uma característica importante da Regressão do Lasso é que ela tende a eliminar completamente os pesos das características menos importantes (ou seja, ajustá-las para zero).

- Em outras palavras, Lasso Regression executa automaticamente a seleção de variáveis/features e exibe um modelo esparso (ou seja, com poucas variáveis não tendo pesos zero).

Lasso Regression

Exemplo

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([ 1.53788174])
```

Elastic Net

$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2$$

Função de custo

- Elastic Net é um meio termo entre Ridge Regression e Lasso Regression. O termo de regularização é uma combinação simples dos termos de regularização de Ridge e Lasso, e você pode controlar a proporção de mistura r. Quando r = 0, Elastic Net é equivalente a Ridge Regression, e quando r = 1, é equivalente a Lasso Regression.

Elastic Net

Exemplo

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([ 1.54333232])
```

Logistics Regression

Logistics Regression

- Alguns algoritmos de regressão também podem ser usados para classificação (e vice-versa). Logistic Regression (também chamado Logit Regression) é usado geralmente para estimar a probabilidade que uma instância pertence a uma determinada classe (por exemplo, qual é a probabilidade de que esta transação seja fraudulenta?).

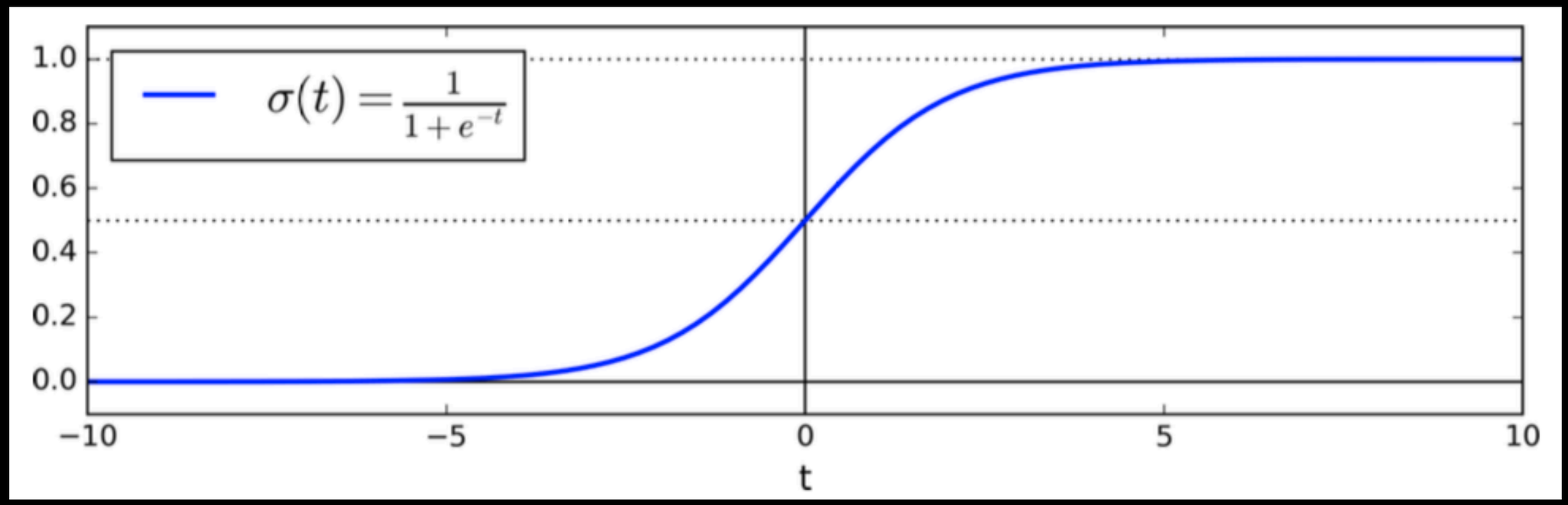
- Se a probabilidade estimada for superior a 50%, então o modelo prevê que a instância pertence a essa classe (chamada classe positiva, rotulada "1"), ou então ela prevê que não (isto é, ela pertence à classe negativa , rotulado como "0"). Isso torna um classificador binário.

Logistics Regression

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T \cdot \mathbf{x}\right)$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$$



Logistics Regression

$$c(\theta) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1, \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0. \end{cases}$$

Função custo de uma instância

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

Função custo de todas instâncias

$$\frac{\partial}{\partial\theta_j}\mathrm{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T \cdot \mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Derivada da função custo — usada na descida de gradiente

Logistics Regression

Exemplo

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> list(iris.keys())
['data', 'target_names', 'feature_names', 'target', 'DESCR']
>>> X = iris["data"][:, 3:]  # petal width
>>> y = (iris["target"] == 2).astype(np.int)  # 1 if Iris-Virginica, else 0
```

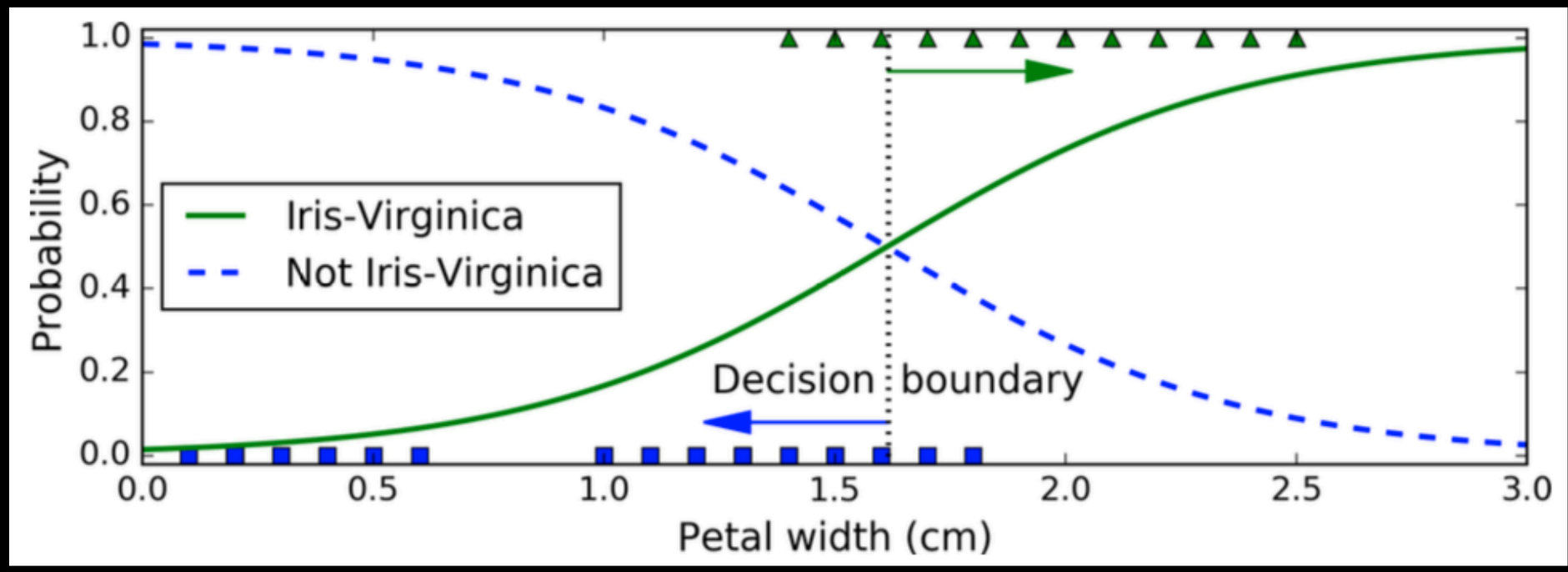
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X, y)
```

```
>>> log_reg.predict([[1.7], [1.5]])
array([1, 0])
```

Logistics Regression

Exemplo

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-Virginica")
```



Logistics Regression

- Assim como os outros modelos lineares, os modelos de Regressão Logística podem ser regularizados usando penalidades de l1 ou l2. O Scitkit-Learn adiciona uma penalidade de l2 por padrão/default.

- O hiperparâmetro que controla a força de regularização de um modelo Scikit-Learn LogisticRegression não é alfa (como em outros modelos lineares), mas é inverso: C. Quanto maior o valor de C, menos o modelo é regularizado.

Aprendizagem Não Supervisionada: K-Means Clustering

K-Means Clustering

- Como modelagem não supervisionada, este modelo procura por estrutura ou grupos num conjunto de dados sem labels.

- O algoritmo k-means busca um número predeterminado de clusters dentro de um conjunto de dados multidimensional não marcado, sem label. Ele consegue isso usando uma concepção simples de como é o agrupamento ótimo:

- O "centro do cluster" é a média aritmética de todos os pontos pertencentes ao cluster.

- Cada ponto de um cluster está mais próximo do seu próprio centro de cluster do que para outros centros de cluster.

K-Means Clustering

- Como ele encontra os clusters:
- 1. Adivinhe alguns centros de cluster
- 2. Repita até convergir o seguinte:
  - 2a. Expectation-Step/Passo: atribua pontos ao centro de cluster mais próximo.
  - 2b. Maximization-Step/Passo: configure os centros de cluster pela média.

K-Means Clustering

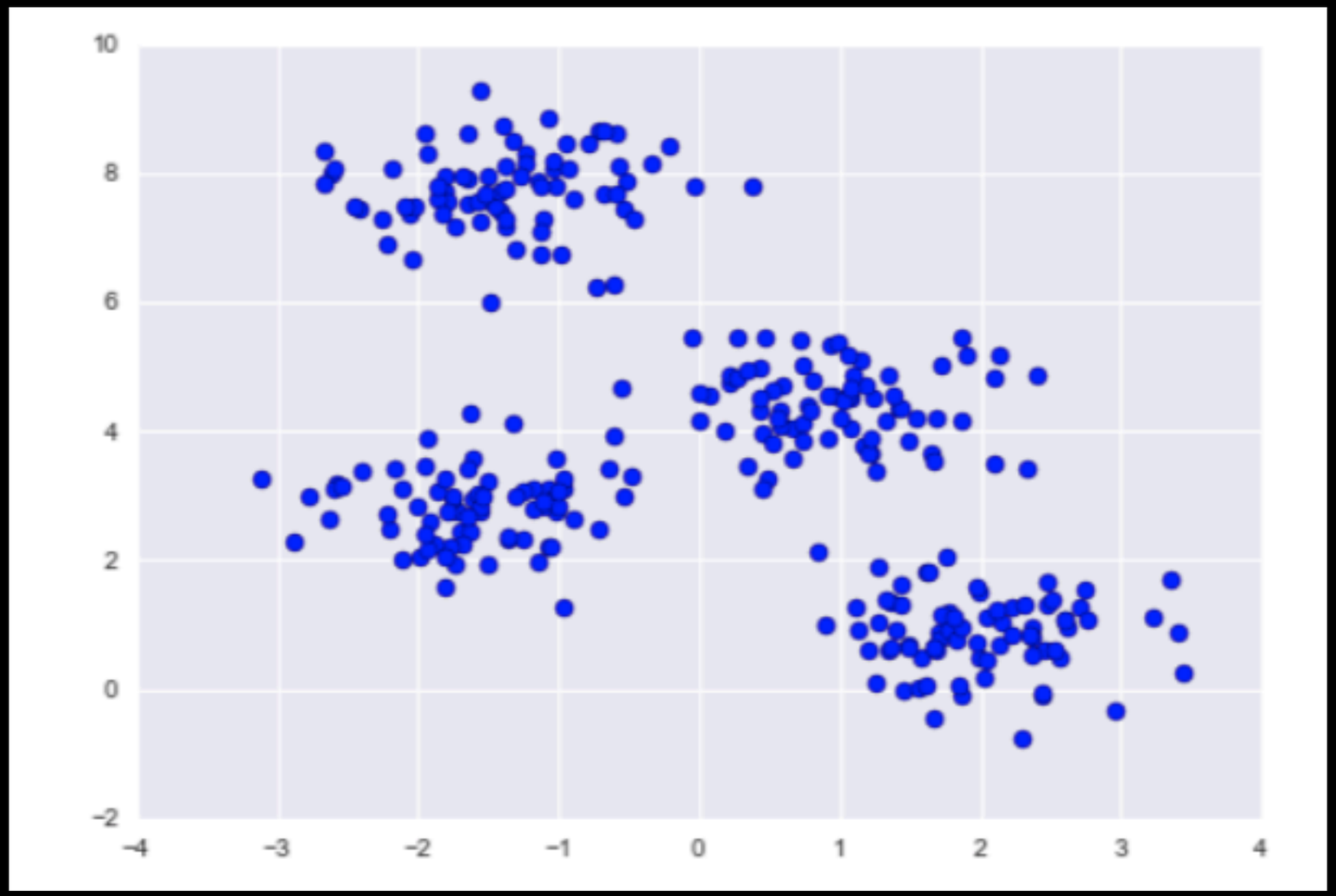
Exemplo

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()  # for plot styling
import numpy as np
```

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

K-Means Clustering

Exemplo



K-Means Clustering

Exemplo

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

K-Means Clustering

Exemplo