

# Agenda

## **UNIDADE 4: Otimização e Desempenho**

- 4.1. Otimização de hiperparâmetros e regularização
  - 4.1.1. Técnicas de avaliação de modelos
    - 4.1.1.1. Overfitting e underfitting
    - 4.1.1.2. Interpretabilidade dos modelos
    - 4.1.1.3. Questões éticas e de viés
  - 4.1.2. Validação cruzada
  - 4.1.3. GridSearchCV
  - 4.1.4. Avaliação de métricas

# Variância e Viés

- VIÉS

- A incapacidade de um modelo de capturar a verdadeira relação entre variáveis e o objeto a ser predito.
- Viés alto => modelo não está aprendendo nada. Presta muito pouca atenção aos dados de treinamento e simplifica demais o modelo.
- Viés pequeno => modelo ajustado demais aos dados de treinamento

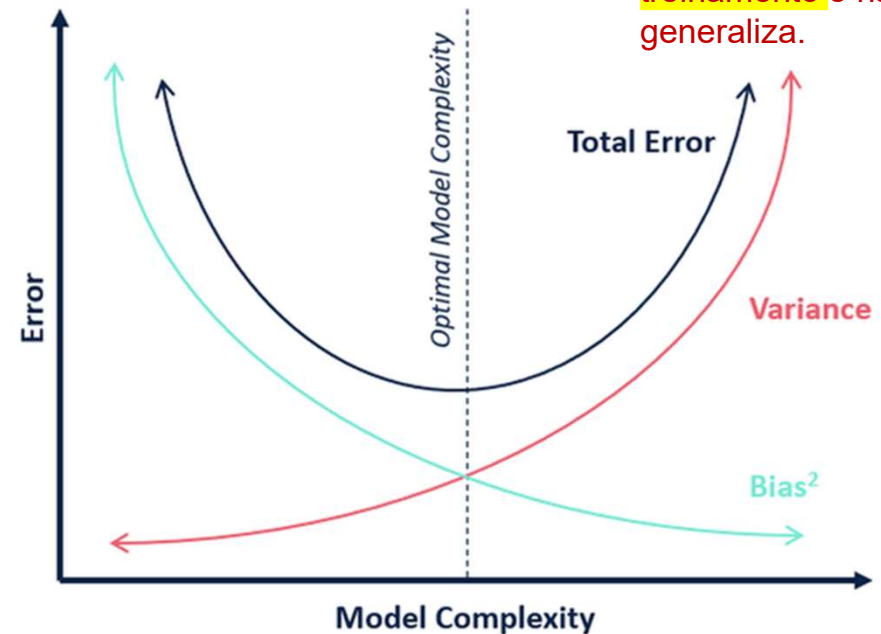
- VARIÂNCIA

- A variância é a sensibilidade de um modelo ao ser usado com outros datasets diferentes do treinamento.
- Alta variância => presta muita atenção aos dados de treinamento e não generaliza

- ERRO TOTAL DO MODELO

- Soma do viés (bias) com a variância

Presta muito pouca atenção aos dados de treinamento e simplifica demais o modelo.

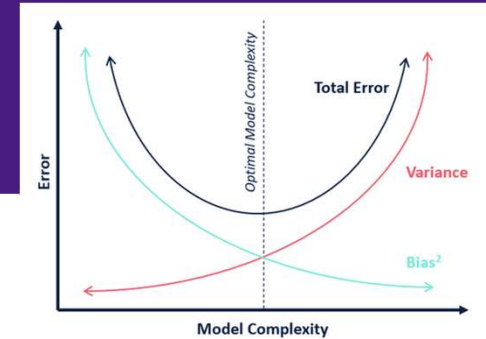


Presta muita atenção aos dados de treinamento e não generaliza.

Alto viés, baixa variância (underfitting)

Baixo viés, alta variância (overfitting)

# Overfitting e underfitting

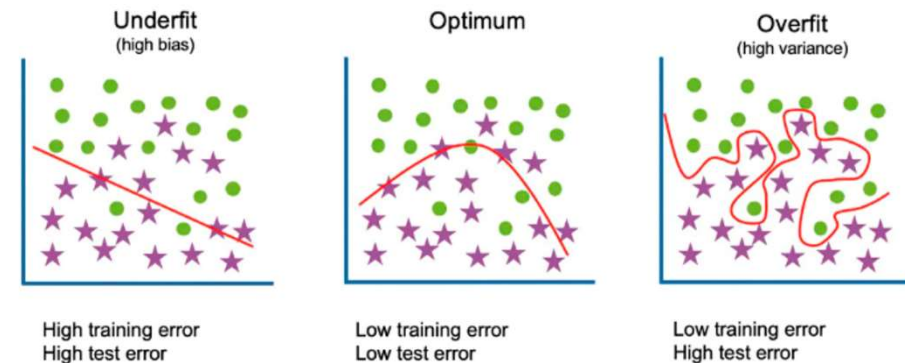


- **Overfitting (sobreajuste) - variância**

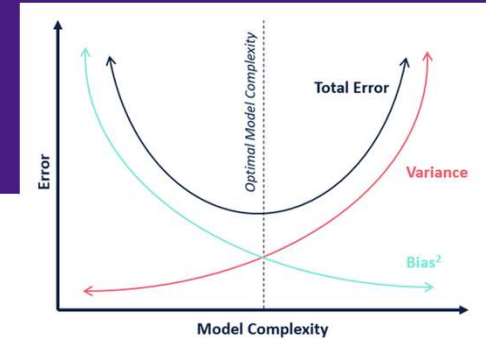
- Modelo  **muito treinado**  com  **baixa capacidade de generalização**
- Modelo preditivo tem  **excelente desempenho nos dados de treinamento** , mas funciona muito  **mal como preditor**

- Como resolver?

- **Aumentar o conjunto de dados**  de treinamento
- **Parar de treinar**  o modelo no tempo adequado
- Usar  **técnicas de regularização**
- **Simplificar**  o modelo



# Overfitting e underfitting

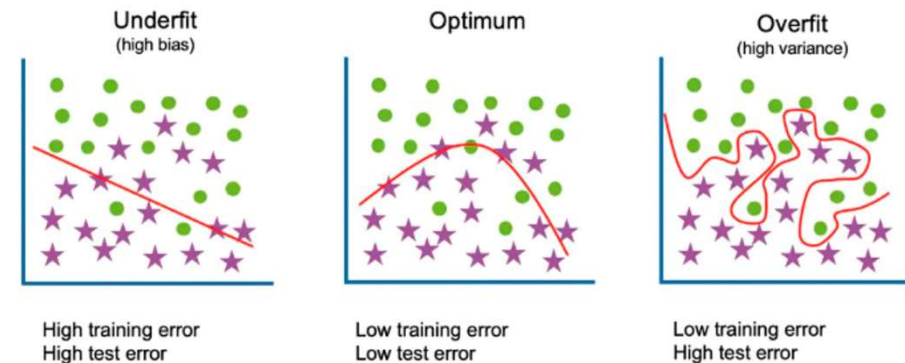


- **Underfitting (subajuste) - viés**

- Modelo **muito simples** ou pouco treinado, **baixa capacidade de generalização**
- Desempenho ruim já nos dados de treinamento
- Modelo preditivo **funciona mal** até mesmo para o conjunto de **dados de treinamento**

- Como resolver?

- **Seleção de variáveis (*feature selection*)**
- **Aumentar a duração do treinamento**
- **Diminuir a regularização**



# Regularização de dados

- Minimizar o **overfitting** nos modelos de aprendizado de máquina.
- Reduzir a influência do ruído no modelo.
- Introduz uma **penalidade para coeficientes elevados**
  - **Somamos um parâmetro de regularização** no cálculo da função custo.
  - **Cálculo do fator: regularização L1 e L2**
    - **Regularização L1 (Lasso Regression):** Aplica uma **penalidade proporcional à soma dos valores absolutos** dos coeficientes.
    - **Regularização L2 (Ridge Regression):** Aplica uma **penalidade proporcional à soma dos quadrados** dos coeficientes.

Ridge Regression (L2 Regularization):

```
python Copiar código

from sklearn.linear_model import Ridge
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# Gerando um conjunto de dados de exemplo
X, y = make_regression(n_samples=100, n_features=20, noise=0.1, random_state=42)

# Dividindo o conjunto de dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Definindo o modelo Ridge com regularização L2
ridge = Ridge(alpha=1.0)

# Treinando o modelo
ridge.fit(X_train, y_train)

# Avaliando o modelo
print("Coeficientes:", ridge.coef_)
print("Intercepto:", ridge.intercept_)
print("Score de treino:", ridge.score(X_train, y_train))
print("Score de teste:", ridge.score(X_test, y_test))
```

L1 Regularization

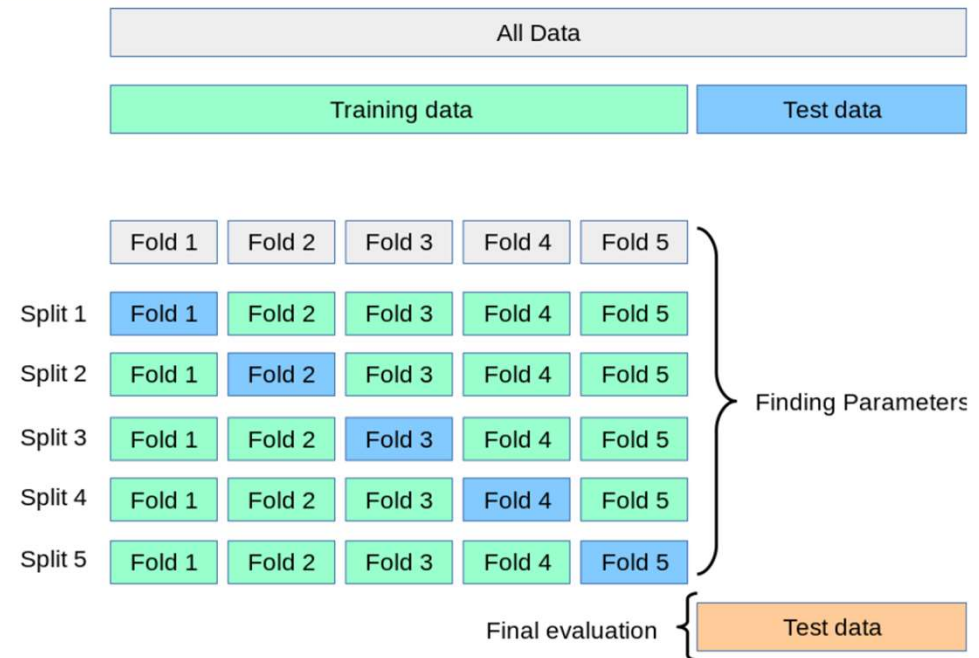
$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^M W_j^2}_{\text{Regularization Term}}$$

# Validação cruzada

- Garantir que o modelo generaliza bem para dados não vistos, ao invés de simplesmente memorizar o conjunto de treino.
- Estimativa confiável do desempenho do modelo durante o treinamento
- Cria k partições de dados:
  - k-1 para treinamento
  - k-ésima para teste,
- Permutando as partições, para treinar k modelos e retornar o desempenho médio



Exemplo de Código: Validação Cruzada com `scikit-learn`

```
python Copiar código

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

# Carregando o conjunto de dados Iris
iris = load_iris()
X, y = iris.data, iris.target

# Definindo o modelo
model = RandomForestClassifier(n_estimators=100)

# Executando a validação cruzada com k=5
scores = cross_val_score(model, X, y, cv=5)

# Imprimindo os resultados
print("Scores de cada iteração:", scores)
print("Desempenho médio:", scores.mean())
print("Desvio padrão:", scores.std())
```

# Parâmetros e Hiperparâmetros

- Parâmetro do modelo
  - variável de que é interna ao modelo;
  - valor pode ser estimado a partir de dados;
  - requeridos pelo modelo para fazer previsões;
  - estimados ou aprendidos a partir de dados;
  - salvos como parte do modelo aprendido.
- Exemplo:
  - os pesos em uma rede neural artificial;
  - os coeficientes em uma regressão linear.

Exemplo: Coeficientes em uma Regressão Linear

```
python Copiar código

from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Gerando um conjunto de dados de exemplo
X, y = make_regression(n_samples=100, n_features=1, noise=0.1)

# Criando e treinando o modelo de Regressão Linear
model = LinearRegression()
model.fit(X, y)

# Parâmetros do modelo: coeficientes e intercepto
print("Coeficiente:", model.coef_)
print("Intercepto:", model.intercept_)

# Fazendo previsões
predictions = model.predict(X)
```

# Parâmetros e Hiperparâmetros

- Hiperparâmetro de um modelo
  - configuração externa ao modelo;
  - valor não pode ser estimado a partir de dados;
  - especificados pelo Cientista de Dados;
  - ajustados para um determinado problema.
- Alguns exemplos de hiperparâmetros modelo incluem:
  - A taxa de aprendizado para treinar uma rede neural.
  - k em k-vizinhos mais próximos.

Exemplo: k em k-Vizinhos Mais Próximos

```
python Copiar código


from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Carregando o conjunto de dados Iris
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2,

# Definindo o hiperparâmetro k
k = 3

# Criando e treinando o modelo KNN
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

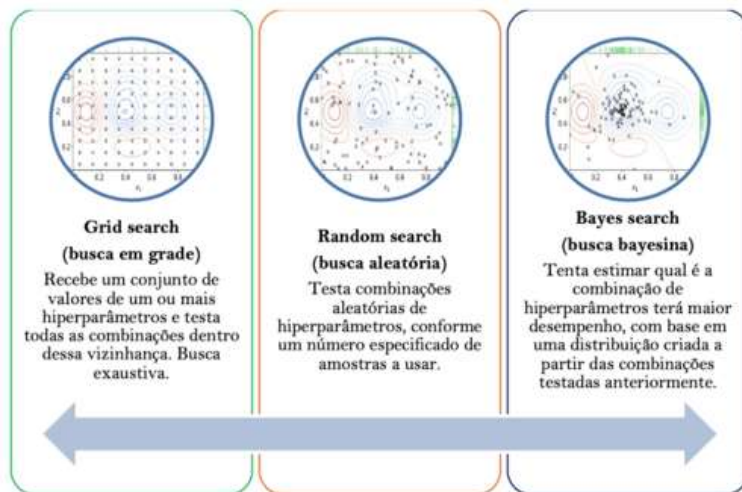
# Fazendo previsões
predictions = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```





# Otimização de Hiperparâmetros

- Processo de **escolher um conjunto ideal de hiperparâmetros** para um determinado algoritmo de aprendizado de máquina.
- **Combinação de configurações que resulta no melhor desempenho** do modelo nos dados disponíveis.



Esquema 15 – Otimização de hiperparâmetros.

Explora exaustivamente uma grade de combinações de hiperparâmetros.

```
python Copiar código

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

# Carregando o conjunto de dados Iris
iris = load_iris()
X, y = iris.data, iris.target

# Definindo o modelo
model = RandomForestClassifier()

# Definindo a grade de hiperparâmetros
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Configurando o GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accu

# Executando o Grid Search
grid_search.fit(X, y)

# Melhor combinação de hiperparâmetros
print("Melhores hiperparâmetros:", grid_search.best_params_)
```

