# K-Fold

November 18, 2025

## 0.1 O que é Desempenho do Modelo e sua Necessidade?

A avaliação do desempenho de modelos de aprendizado de máquina é semelhante à avaliação de notas na escola ou faculdade, onde verificamos se atendemos aos critérios de elegibilidade para os melhores cursos, entrevistas de emprego ou exames competitivos. Um bom desempenho indica que o candidato é consistentemente bom. O mesmo é esperado de um modelo de aprendizado de máquina, que deve alcançar resultados precisos em previsões, classificações e outros problemas.

## 0.2 O que é a Precisão do Modelo e Desempenho?

A precisão é apenas um número que ajuda a entender melhor um problema baseado em previsão, corrigindo as previsões feitas pelo modelo com os registros disponíveis. Portanto, precisamos treinar o modelo com diferentes combinações de dados.

Como mencionei anteriormente, temos diferentes técnicas para avaliar, nas quais a Validação Cruzada ou Validação Cruzada K-Fold é a melhor e mais fácil de entender. É simples por natureza e envolve uma técnica típica de reamostragem, sem substituição dos dados. E podemos entender e visualizar facilmente ao implementar.

## 0.3 Validação Cruzada K-Fold

Em cada conjunto (fold), o treinamento e o teste são realizados exatamente uma vez durante todo o processo. Isso nos ajuda a evitar o overfitting. Quando um modelo é treinado usando todos os dados de uma só vez, ele pode apresentar a melhor precisão. A validação cruzada K-Fold nos ajuda a construir um modelo mais generalizado.

Para realizar a Validação Cruzada K-Fold, precisamos dividir o conjunto de dados em três partes: Treinamento, Teste e Validação, considerando o volume de dados.

Os conjuntos de dados de Teste e Treinamento suportam a construção do modelo e a avaliação de hiperparâmetros.

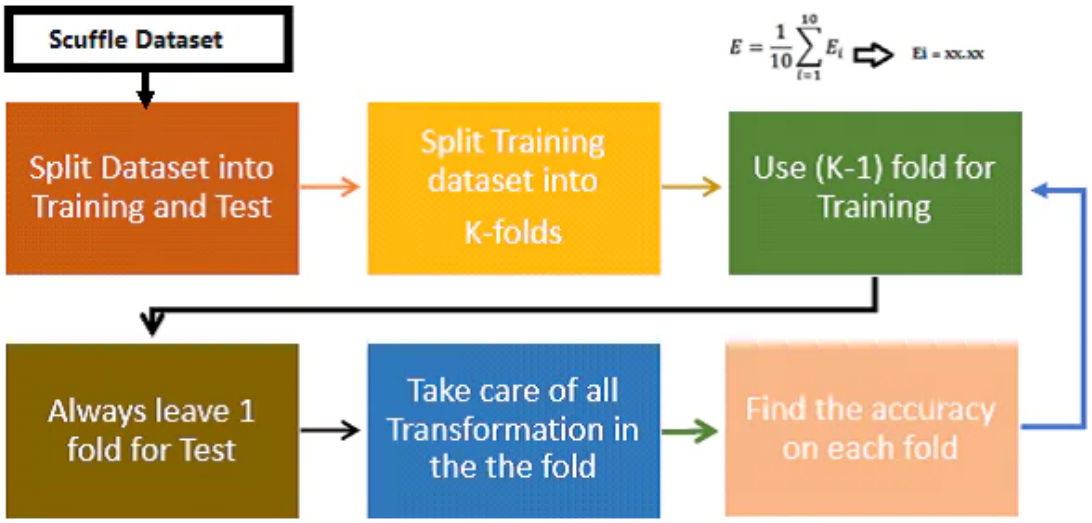
O modelo é validado várias vezes com base no valor atribuído como parâmetro, chamado K, que deve ser um número inteiro.

Simplificando, com base no valor de K, o conjunto de dados é dividido, e o treinamento/teste é realizado de forma sequencial igual ao número de vezes definido por K.

## 0.4 Ciclo de Vida da Validação Cruzada K-Fold

[1]:
```
from IPython.display import Image
Image(filename='2.png')
```

[1]:




Vamos considerar um valor de K generalizado. Se K=5, significa que o conjunto de dados é dividido em 5 folds e o treinamento e teste são realizados. Durante cada execução, um fold é usado para teste e os demais para treinamento. A representação pictórica abaixo ilustra o fluxo do tamanho definido pelo fold.

[2]:
```
Image(filename='3.png')
```

[2]:

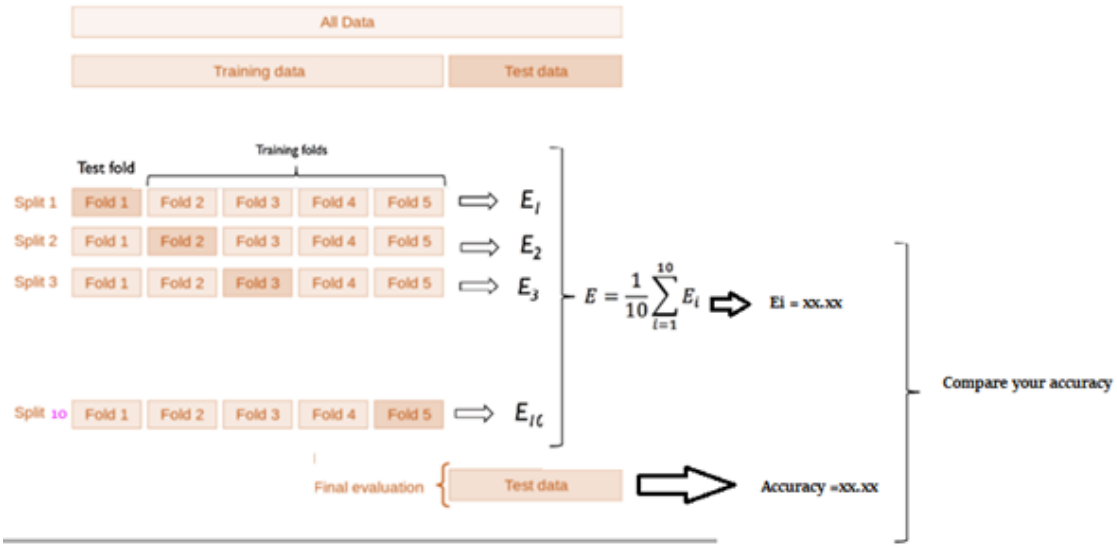

Exemplo de Validação Cruzada K-Fold (KFCV)

Em cada iteração, cada ponto de dados é usado uma vez no conjunto de teste e K-1 vezes no treinamento. Durante a iteração completa, pelo menos uma vez, um fold será usado para teste e os demais para treinamento.

No exemplo acima, 5 folds são usados para teste e 20 para treinamento. Em cada iteração, obtemos uma pontuação de precisão, somamos todas e calculamos a média. Isso nos ajuda a entender como os dados estão distribuídos de forma consistente e a decidir se o modelo está pronto para produção ou não.

[3]:
```
Image(filename='4.png')
```

[3]:



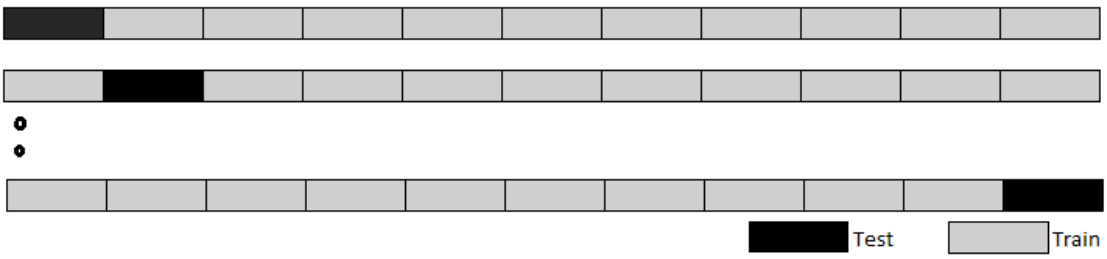

## 0.5 Regras Básicas Associadas ao K-Fold

Agora, vamos discutir algumas regras básicas ao usar o K-Fold:

- K deve ser sempre >= 2 e igual ao número de registros (LOOCV).
  - Se K=2, teremos apenas 2 iterações.
  - Se K for igual ao número de registros no conjunto de dados, teremos 1 para teste e n-1 para treinamento.
- O valor otimizado para K = 10 é usado com dados de bom tamanho (comumente usado).
- Se o valor de K for muito grande, isso levará a menor variância no conjunto de treinamento e limitará a diferença de desempenho do modelo nas iterações.
- O número de folds (conjuntos) é inversamente proporcional ao tamanho do conjunto de dados, ou seja, se o conjunto de dados for muito pequeno, o número de folds pode aumentar.
- Valores maiores de K aumentam o tempo de execução do processo de validação cruzada.

[4]:
```
Image(filename='5.png')
```

[4]:



Lembre-se de usar a Validação Cruzada K-Fold para os seguintes propósitos no aprendizado de máquina:

- Seleção de modelos
- Ajuste de parâmetros
- Seleção de características

Até agora, discutimos o K-Fold e sua implementação. Vamos fazer um exemplo prático agora.

### 0.5.1 Exemplo Básico

Vamos criar um array simples, definir o tamanho de K como 5 e dividir meu array. Usando um loop simples, imprimirei as partes de Treinamento e Teste. Aqui podemos ver claramente que os pontos de dados nos buckets de Treinamento e Teste são únicos em cada ciclo.

Você pode ver os arrays de Treinamento e Teste e como o array foi dividido em cada iteração.

Vamos fazer isso com um conjunto de dados.

Seleção de Modelos usando K-Fold

[5]:
```
# Importando bibliotecas necessárias
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Carregando o conjunto de dados (dígitos manuscritos - open source)
digits = load_digits()

# Dividindo o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,␣
 ↪test_size=0.3)

# Aplicando diferentes algoritmos
```

## 0.6 Regressão Logística

Usando liblinear, que é da categoria “Large Linear Classification”. Ele usa um Algoritmo de Descida Coordenada, que minimiza uma função multivariada resolvendo problemas univariados de otimização durante o loop.

[6]:
```
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

```
print("Score:", lr.score(X_test, y_test))
```

```
Score: 0.9555555555555556
```

### 0.7 SVC

Usando gamma como um parâmetro para a perspectiva não linear dos hiperplanos. O valor de gamma tenta ajustar o conjunto de dados de treinamento e usa 1/n_features.

[9]:
```
svm = SVC(gamma='auto')
svm.fit(X_train, y_train)
print("Score:", svm.score(X_test, y_test))
```

```
Score: 0.3388888888888889
```

### 0.8 Random Forest

Para o Random Forest, estou atribuindo n_estimators como 40.

[8]:
```
rf = RandomForestClassifier(n_estimators=40)
rf.fit(X_train, y_train)
print("Score:", rf.score(X_test, y_test))
```

```
Score: 0.9666666666666667
```

Os resultados acima mostram que a Regressão Logística e o Random Forest têm um desempenho comparativamente melhor do que o SVM.

Agora, vamos usar a função cross_val_score e obter as pontuações, passando diferentes algoritmos com o conjunto de dados e cv.

[10]:
```
from sklearn.model_selection import cross_val_score

# Definindo LogisticRegression e CV=3
score_lr = cross_val_score(LogisticRegression(solver='liblinear',␣
 ↪multi_class='ovr'), digits.data, digits.target, cv=3)
print(score_lr)
print("Avg:", np.average(score_lr))
```

```
[0.89482471 0.95325543 0.90984975]
Avg: 0.9193099610461881
```

[11]:
```
# Definindo SVM e CV=3
score_svm = cross_val_score(SVC(gamma='auto'), digits.data, digits.target, cv=3)
print(score_svm)
print("Avg:", np.average(score_svm))
```

```
[0.38063439 0.41068447 0.51252087]
Avg: 0.4346132442960489
```

[12]:
```
# Definindo Random Forest e CV=3
score_rf = cross_val_score(RandomForestClassifier(n_estimators=40), digits.
 ↪data, digits.target, cv=3)
print(score_rf)
print("Avg:", np.average(score_rf))
```

```
[0.92821369 0.95158598 0.92153589]
Avg: 0.9337785197551475
```

| Algoritmo | Antes do K-Fold | Depois do K-Fold (Média) |
|---|---|---|
| Regressão Logística | 97% | 91% |
| SVM | 38% | 43% |
| Random Forest | 96% | 93% |

Com base na tabela acima, vamos usar o Random Forest para este conjunto de dados. No entanto, precisamos monitorar o desempenho do modelo com base na variação dos dados e, conforme o caso de uso do negócio mudar, revisitar o modelo e reimplantá-lo.

## 0.9 Ajuste de Parâmetros Usando K-Fold

Vamos considerar o RandomForestClassifier para esta análise. O parâmetro que vamos ajustar é o n_estimators e o valor de CV será 10, que é comumente utilizado.

[13]:
```
# Importando bibliotecas necessárias
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Ajustando o parâmetro n_estimators para 5 e usando CV=10
scores1 = cross_val_score(RandomForestClassifier(n_estimators=5), digits.data,␣
 ↪digits.target, cv=10)
print("Avg Score for Estimators=5 and CV=10:", np.average(scores1))

# Ajustando o parâmetro n_estimators para 20 e usando CV=10
scores2 = cross_val_score(RandomForestClassifier(n_estimators=20), digits.data,␣
 ↪digits.target, cv=10)
print("Avg Score for Estimators=20 and CV=10:", np.average(scores2))

# Ajustando o parâmetro n_estimators para 30 e usando CV=10
scores3 = cross_val_score(RandomForestClassifier(n_estimators=30), digits.data,␣
 ↪digits.target, cv=10)
print("Avg Score for Estimators=30 and CV=10:", np.average(scores3))


# Ajustando o parâmetro n_estimators para 40 e usando CV=10
scores4 = cross_val_score(RandomForestClassifier(n_estimators=40), digits.data,␣
 ↪digits.target, cv=10)
```

```
print("Avg Score for Estimators=40 and CV=10:", np.average(scores4))
```

```
Avg Score for Estimators=5 and CV=10: 0.872554314090627
Avg Score for Estimators=20 and CV=10: 0.9348727498448168
Avg Score for Estimators=30 and CV=10: 0.9382371198013656
Avg Score for Estimators=40 and CV=10: 0.9465518311607696
```

| n_estimators | Média da Pontuação (%) |
|---|---|
| 5 | 87.36 |
| 20 | 93.33 |
| 30 | 94.87 |
| 40 | 94.82 |

Com base na observação acima, vamos usar n_estimators=30.

## 0.10 K-Fold em Forma Visual

A representação visual é sempre a melhor evidência para qualquer dado que está localizado nos eixos.

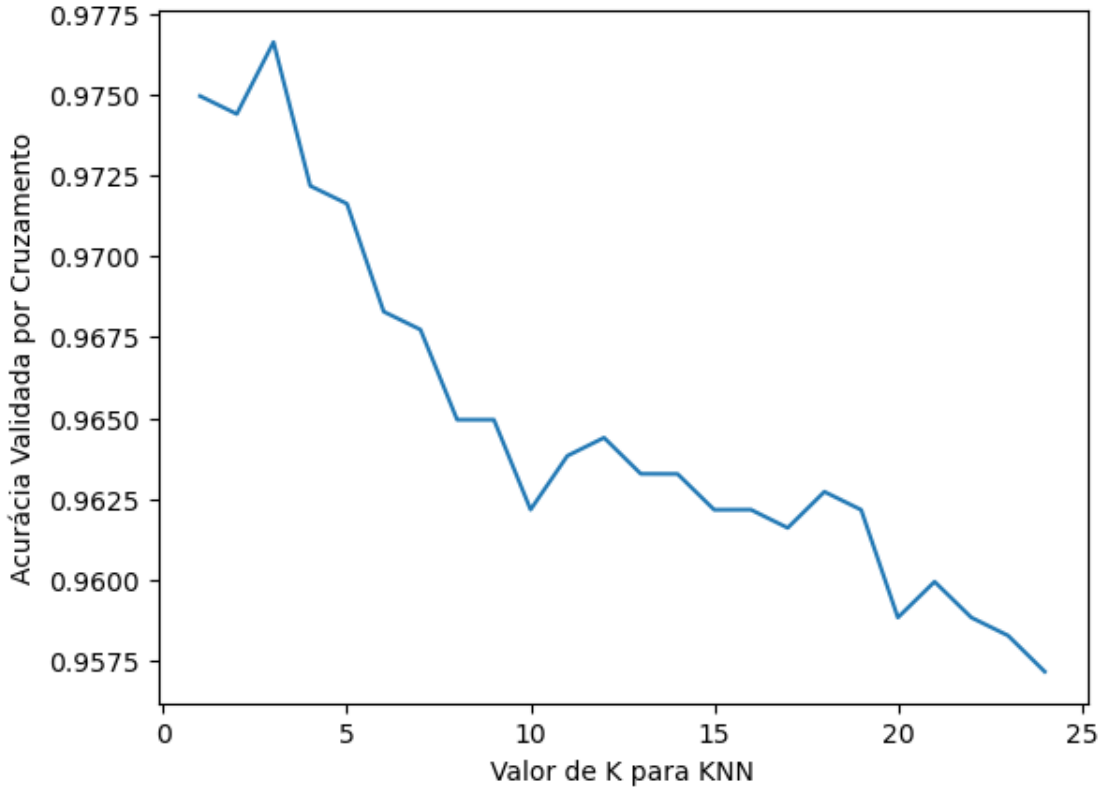
[15]:
```
# Importando bibliotecas necessárias
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

# Definindo o classificador KNN com n_neighbors=5
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn, digits.data, digits.target, cv=10,␣
 ↪scoring='accuracy')
print(scores.mean())


# Testando diferentes valores de K
k_range = list(range(1, 25))
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, digits.data, digits.target, cv=10,␣
 ↪scoring='accuracy')
    k_scores.append(scores.mean())
print(k_scores)

# Plotando os resultados
plt.plot(k_range, k_scores)
plt.xlabel('Valor de K para KNN')
plt.ylabel('Acurácia Validada por Cruzamento')
plt.show()
```

```
0.9716294227188081
[0.9749627560521414, 0.974407200496586, 0.9766325263811299, 0.9721818746120421,
0.9716294227188081, 0.9682898820608317, 0.9677343265052762, 0.9649441340782122,
0.9649441340782122, 0.9621632526381129, 0.963826815642458, 0.9643885785226567,
0.9632712600869026, 0.9632681564245809, 0.9621570453134698, 0.9621601489757914,
0.9615983860955927, 0.9627157045313469, 0.9621570453134698, 0.9588237119801365,
0.9599317194289261, 0.9588237119801365, 0.9582650527622594, 0.9571477343265051]
```




[16]:
```
from sklearn.model_selection import KFold

import numpy as np

data=np.array([5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100])
kfold=KFold(5,shuffle = True)
for train,test in kfold.split(data):
  print("Train: %s,Test: %s"%(data[train],data[test]))
```

```
Train: [  5  10  15  25  30  35  40  45  55  65  75  80  85  90  95 100],Test:
[20 50 60 70]
Train: [ 5 10 20 25 40 45 50 55 60 65 70 75 80 85 90 95],Test: [ 15  30  35 100]
Train: [  5  10  15  20  30  35  40  45  50  55  60  70  80  85  95 100],Test:
[25 65 75 90]
```

```
Train: [ 10  15  20  25  30  35  40  50  60  65  70  75  80  90  95 100],Test: [
5 45 55 85]
Train: [  5  15  20  25  30  35  45  50  55  60  65  70  75  85  90 100],Test:
[10 40 80 95]
```