Folha de Dicas: Avaliando e Validando Modelos de Aprendizado de Máquina

Métricas e métodos de avaliação de modelos

Nome do Método	Descrição	Sintaxe do Código

classification_report	Gera um relatório com precisão, recall, F1-score e suporte para cada classe em problemas de classificação. Útil para avaliação de modelos. **Hiperparâmetros**: target_names: Lista de rótulos a serem incluídos no relatório. **Prós**: Fornece uma avaliação abrangente de modelos de classificação. **Limitações**: Pode não fornecer informações suficientes para conjuntos de dados desbalanceados.	`from sklearn.metrics import classification_report`
		y_true: Rótulos verdadeiros
		y_pred: Rótulos previstos
		target_names: Lista de nomes das classes alvo
		`report = classification_report(y_true, y_pred, target_names=[“class1”, “class2”])`

confusion_matrix	Calcula uma matriz de confusão para avaliar o desempenho da classificação, mostrando contagens	`from sklearn.metrics import confusion_matrix` **y_true: Rótulos verdadeiros** **y_pred: Rótulos previstos**

	de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos. **Hiperparâmetros**: labels: Lista de rótulos de classe a incluir. **Vantagens**: Essencial para entender os erros de classificação. **Limitações**: Não fornece insights sobre as probabilidades de previsão.	`conf_matrix = confusion_matrix(y_true, y_pred)`

mean_squared_error	Calcula o erro quadrático médio (MSE), uma métrica comum para modelos de regressão. Valores mais baixos indicam melhor desempenho. **Hiperparâmetros**: sample_weight: Pesos a serem aplicados a cada amostra. **Prós**: Métrica simples e amplamente utilizada. **Limitações**: Sensível a outliers, pois grandes erros são elevados ao quadrado.	`from sklearn.metrics import mean_squared_error`
		y_true: Valores verdadeiros
		y_pred: Valores previstos
		sample_weight: Opcional, array de pesos amostrais
		`mse = mean_squared_error(y_true, y_pred)`

root_mean_squared_error	Calcula o erro quadrático médio da raiz (RMSE), que é a raiz quadrada do MSE. O RMSE fornece resultados mais interpretáveis, pois está nas mesmas unidades que o alvo. **Hiperparâmetros**: sample_weight: Pesos a serem aplicados a cada amostra. **Vantagens**: Mais interpretável do que o MSE.	`from sklearn.metrics import root_mean_squared_error`
		y_true: Valores verdadeiros
		y_pred: Valores previstos
		sample_weight: Opcional, array de pesos das amostras
		`rmse = root_mean_squared_error(y_true, y_pred)`

	Limitações: Assim como o MSE, pode ser sensível a grandes erros e outliers.	

mean_absolute_error	Medida da magnitude média dos erros nas previsões, sem considerar sua direção. Útil para entender o tamanho médio do erro. **Hiperparâmetros**: sample_weight: Pesos das amostras opcionais. **Prós**: Menos sensível a outliers em comparação com o MSE. **Limitações**: Não penaliza erros grandes tanto quanto o MSE ou RMSE.	`from sklearn.metrics import mean_absolute_error`
		y_true: Valores verdadeiros
		y_pred: Valores previstos
		`mae = mean_absolute_error(y_true, y_pred)`

r2_score	Calcula o coeficiente de determinação (R^2), que representa a proporção da variância explicada pelo modelo. Um valor mais alto indica um melhor ajuste. **Prós**: Fornece uma indicação clara do desempenho do modelo. **Limitações**: Não representa sempre a qualidade do modelo, especialmente para modelos não lineares.	`from sklearn.metrics import r2_score`
		y_true: Valores verdadeiros
		y_pred: Valores previstos
		`r2 = r2_score(y_true, y_pred)`

silhouette_score	Mede a qualidade da agrupamento avaliando a coesão dentro dos grupos e a separação entre os grupos. Pontuações mais altas indicam um melhor agrupamento. **Hiperparâmetros**: métrica: Métrica de distância a ser utilizada.	`from sklearn.metrics import silhouette_score`
		X: Dados utilizados na agrupamento
		rótulos: Rótulos de cluster para cada amostra
		`score = silhouette_score(X, labels, metric='euclidean')`

	Prós: Útil para validar o desempenho do agrupamento. **Limitações**: Sensível a outliers e à escolha da métrica de distância.	

silhouette_samples	Fornece pontuações de silhueta para cada amostra individual, indicando quão bem ela se encaixa em seu cluster designado. **Hipertensões**: métrica: Métrica de distância a ser utilizada. **Prós**: Oferece uma visão detalhada da qualidade de agrupamento de cada amostra. **Limitações**: As mesmas do silhouette_score; sensível a outliers e à métrica de distância.	`from sklearn.metrics import silhouette_samples`
		X: Dados usados na clusterização
		rótulos: Rótulos de cluster para cada amostra
		`samples = silhouette_samples(X, labels, metric='euclidean')`

davies_bouldin_score	Medida da razão média de similaridade de cada cluster com o cluster mais semelhante. Valores mais baixos indicam uma melhor agrupamento. **Prós**: Fornece uma avaliação de agrupamento simples e eficaz. **Limitações**: Pode não funcionar bem com clusters altamente desbalanceados.	`from sklearn.metrics import davies_bouldin_score`
		X: Dados utilizados na agrupamento
		rótulos: Rótulos de cluster para cada amostra
		`db_score = davies_bouldin_score(X, labels)`

Voronoi	Calcula o diagrama de Voronoi, que partitiona o espaço com base no vizinho mais próximo. **Prós**: Útil para análise espacial e clustering. **Limitações**: Limitado a casos de uso que envolvem a	`from scipy.spatial import Voronoi`
		pontos: Coordenadas para o diagrama de Voronoi
		`vor = Voronoi(pontos)`

	partição espacial de dados.	

voronoi_plot_2d	Plota o diagrama de Voronoi em 2D para visualizar os resultados de agrupamento. **Hiperparâmetros**: show_vertices: Se deve exibir os vértices. **Prós**: Ótimo para visualizar agrupamentos espaciais. **Limitações**: Limitado a espaços 2D e grandes conjuntos de dados podem causar problemas de desempenho.	`from scipy.spatial import voronoi_plot_2d`
		vor: Objeto do diagrama de Voronoi
		`voronoi_plot_2d(vor, show_vertices=True)`

matplotlib.patches.Patch	Cria formas personalizadas, como retângulos, círculos ou elipses, para adicionar a gráficos. **Hiperparâmetros**: cor: Preenche a cor da forma. **Prós**: Versátil para personalização visual. **Limitações**: Pode não suportar todas as formas ou personalizações complexas.	`import matplotlib.patches as patches`
		Crie um retângulo com largura, altura e posição especificadas
		`rectangle = patches.Rectangle((0, 0), 1, 1, color='blue')`

explained_variance_score	Meça a proporção da variância explicada pelas previsões do modelo. Uma pontuação mais alta indica melhor desempenho. **Prós**: Ajuda na avaliação do ajuste de modelos de regressão. **Limitações**: Não é adequado para tarefas de classificação.	`from sklearn.metrics import explained_variance_score`
		y_true: Valores verdadeiros
		y_pred: Valores previstos
		`ev_score = explained_variance_score(y_true, y_pred)`

Regressão Ridge	Executa regressão ridge (regularização	`from sklearn.linear_model import Ridge`

	L2) para evitar overfitting penalizando coeficientes grandes. **Hiperparâmetros**: alpha: Força da regularização. **Prós**: Ajuda a reduzir o overfitting em modelos de regressão. **Limitações**: Pode não funcionar bem com dados esparsos.	**alpha: Força de regularização (valores maiores indicam regularização mais f** `ridge = Ridge(alpha=1.0)`

Regressão Lasso	Realiza regressão lasso (regularização L1), que incentiva a esparsidade ao penalizar o valor absoluto dos coeficientes. **Hiperparâmetros**: alpha: Força de regularização. **Prós**: Incentiva soluções esparsas, útil para seleção de características. **Limitações**: Pode ter dificuldades com multicolinearidade.	`from sklearn.linear_model import Lasso`
		alpha: Força da regularização (valores maiores indicam regularização mais f
		`lasso = Lasso(alpha=0.1)`

Pipeline	Encadeia múltiplas etapas de pré-processamento e modelagem em um único objeto, garantindo um fluxo de trabalho eficiente. **Prós**: Simplifica o código, garante reprodutibilidade. **Limitações**: Pode não funcionar bem com pipelines complexos que requerem configurações dinâmicas.	`from sklearn.pipeline import Pipeline`
		etapas: Lista de tuplas com nome e estimador/transformador
		`pipeline = Pipeline(steps=[('scaler', StandardScaler()), ('model', Ridge(alpha=1.0))])`

GridSearchCV	Executa uma busca exaustiva sobre uma grade de parâmetros especificada para encontrar a melhor configuração do modelo. **Hiperparâmetros**: param_grid:	`from sklearn.model_selection import GridSearchCV`
		estimador: Modelo a ser ajustado
		param_grid: Dicionário com parâmetros para pesquisar
		`grid_search = GridSearchCV(estimator=Ridge(), param_grid={'alpha': [0.1, 1.0, 10.0]})`

Dicionário de grades de parâmetros. **Vantagens**: Garante parâmetros ótimos para o modelo. **Limitações**: Computacionalmente caro para grades grandes.

Estratégias de visualização para avaliação de k-means

Nome do Processo
Descrição Breve
Trecho de Código

Múltiplas execuções do k-means
Executa o agrupamento KMeans várias vezes com diferentes inicializações aleatórias para avaliar a variabilidade nas atribuições de cluster.
Vantagem: Ajuda a visualizar a consistência.
Limitação: Custoso computacionalmente para grandes conjuntos de dados.

```
# Número de execuções para KMeans com diferentes estados aleatórios
n_runs = 4
inertia_values = []

plt.figure(figsize=(12, 12))

# Executa K-Means várias vezes com diferentes estados aleatórios
for i in range(n_runs):
    kmeans = KMeans(n_clusters=4, random_state=None)  # Usa o padrão `n_init`
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

    # Plota o resultado do agrupamento
    plt.subplot(2, 2, i + 1)
    plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='tab10', alpha=0.6, edgecolor='k')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', s=200, marker='x', label='Centroides')
    plt.title(f'Execução de Agrupamento K-Means {i + 1}')
    plt.xlabel('Recurso 1')
```

```
    plt.ylabel('Recurso 2')
    plt.legend()

plt.tight_layout()
plt.show()

# Imprime os valores de inércia
for i, inertia in enumerate(inertia_values, start=1):
    print(f'Execução {i}: Inércia={inertia:.2f}')
```

Método do Cotovelo

Avalia o número ótimo de clusters plotando a inércia (soma dos quadrados dentro do cluster) para diferentes valores de **k**.

Vantagem: Fácil de interpretar.

Limitação: Ponto de cotovelo subjetivo.

```

# Faixa de valores de k para testar
k_values = range(2, 11)

# Armazena métricas de desempenho
inertia_values = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    y_kmeans = kmeans.fit_predict(X)

    # Calcula e armazena métricas
    inertia_values.append(kmeans.inertia_)

# Plota os valores de inércia (Método do Cotovelo)
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.plot(k_values, inertia_values, marker='o')
```

```
plt.title('Método do Cotovelo: Inércia vs. k')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inércia')
```

Método da Silhueta

Determina o número ótimo de clusters avaliando as Pontuações de Silhueta para diferentes valores de **k**.

Vantagem: Considera tanto a coesão quanto a separação.

Limitação: Alta computação para grandes conjuntos de dados.

```
# Faixa de valores de k para testar
k_values = range(2, 11)

# Armazena métricas de desempenho
silhouette_scores = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    y_kmeans = kmeans.fit_predict(X)
    silhouette_scores.append(silhouette_score(X, y_kmeans))

# Plota as Pontuações de Silhueta
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 2)
plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Pontuação de Silhueta vs. k')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Pontuação de Silhueta')
```

Índice de Davies-Bouldin

Avalia o desempenho do agrupamento calculando o DBI para diferentes valores de **k**.
Vantagem: Quantifica a compacidade e separação.
Limitação: Sensível a formas e densidade dos clusters.

```
# Faixa de valores de k para testar
k_values = range(2, 11)

# Armazena métricas de desempenho
davies_bouldin_indices = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    y_kmeans = kmeans.fit_predict(X)
    davies_bouldin_indices.append(davies_bouldin_score(X, y_kmeans))

# Plota o Índice de Davies-Bouldin
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 3)
plt.plot(k_values, davies_bouldin_indices, marker='o')
plt.title('Índice de Davies-Bouldin vs. k')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Índice de Davies-Bouldin')
```

Autores

Jeff Grossman
Abhishek Gagneja