Hoja de trucos: Evaluación y validación de modelos de aprendizaje automático

Métricas y métodos de evaluación del modelo

Nombre del Método	Descripción	Sintaxis del Código

classification_report	Genera un informe con precisión, recuperación, F1-score y soporte para cada clase en problemas de clasificación. Útil para la evaluación del modelo. **Hiperparámetros**: target_names: Lista de etiquetas a incluir en el informe. **Ventajas**: Proporciona una evaluación integral de los modelos de clasificación. **Limitaciones**: Puede no proporcionar suficiente información para conjuntos de datos desbalanceados.	`from sklearn.metrics import classification_report`
		y_true: Etiquetas verdaderas
		y_pred: Etiquetas predichas
		target_names: Lista de nombres de clases objetivo
		`report = classification_report(y_true, y_pred, target_names=["class1", "class2"])`

confusion_matrix	Calcula una matriz de confusión para evaluar el rendimiento de la clasificación, mostrando los conteos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. **Hiperparámetros**: labels: Lista de etiquetas de clase a incluir. **Ventajas**: Esencial para comprender los errores de clasificación. **Limitaciones**: No proporciona información sobre las probabilidades de predicción.	`from sklearn.metrics import confusion_matrix`
		y_true: Etiquetas verdaderas
		y_pred: Etiquetas predichas
		`conf_matrix = confusion_matrix(y_true, y_pred)`

mean_squared_error	Calcula el error cuadrático medio (MSE), una métrica común para modelos de regresión. Valores más bajos indican un mejor rendimiento. **Hiperparámetros**: sample_weight: Pesos a aplicar a cada muestra. **Ventajas**: Métrica simple y ampliamente utilizada. **Limitaciones**: Sensible a los valores atípicos, ya que los errores	`from sklearn.metrics import mean_squared_error`
		y_true: Valores verdaderos
		y_pred: Valores predichos
		sample_weight: Opcional, arreglo de pesos de muestr
		`mse = mean_squared_error(y_true, y_pred)`

	grandes se elevan al cuadrado.	

root_mean_squared_error	Calcula la raíz del error cuadrático medio (RMSE), que es la raíz cuadrada del MSE. RMSE ofrece resultados más interpretables ya que está en las mismas unidades que el objetivo. **Hiperparámetros**: sample_weight: Pesos a aplicar a cada muestra. **Ventajas**: Más interpretable que el MSE. **Limitaciones**: Al igual que el MSE, puede ser sensible a grandes errores y valores atípicos.	`from sklearn.metrics import root_mean_squared_error`
		y_true: Valores verdaderos
		y_pred: Valores predichos
		sample_weight: Opcional, arreglo de pesos de muestr
		`rmse = root_mean_squared_error(y_true, y_pred)`

mean_absolute_error	Mide la magnitud promedio de los errores en las predicciones, sin considerar su dirección. Útil para entender el tamaño del error promedio. **Hiperparámetros**: sample_weight: Pesos de muestra opcionales. **Ventajas**: Menos sensible a los valores atípicos en comparación con MSE. **Limitaciones**: No penaliza los errores grandes tanto como MSE o RMSE.	`from sklearn.metrics import mean_absolute_error`
		y_true: Valores verdaderos
		y_pred: Valores predichos
		`mae = mean_absolute_error(y_true, y_pred)`

r2_score	Calcula el coeficiente de determinación (R^2), que representa la proporción de varianza explicada por el modelo. Un valor más alto indica un mejor ajuste. **Pros**: Proporciona una indicación clara del rendimiento del modelo. **Limitaciones**: No siempre representa la calidad del modelo, especialmente para modelos no lineales.	`from sklearn.metrics import r2_score`
		y_true: Valores verdaderos
		y_pred: Valores predichos
		`r2 = r2_score(y_true, y_pred)`

silhouette_score	Mide la calidad de la agrupación evaluando la cohesión dentro de los grupos y la separación entre ellos. Puntuaciones más altas indican una mejor agrupación. **Hiperparámetros**: métrica: Métrica de	`from sklearn.metrics import silhouette_score`
		X: Datos utilizados en la agrupación
		etiquetas: Etiquetas de clúster para cada muestra
		`score = silhouette_score(X, labels, metric='euclidean')`

	distancia a utilizar. **Ventajas**: Útil para validar el rendimiento de la agrupación. **Limitaciones**: Sensible a los valores atípicos y a la elección de la métrica de distancia.	

silhouette_samples	Proporciona puntuaciones de silueta para cada muestra individual, indicando qué tan bien se ajusta a su clúster asignado. **Hiperparámetros**: métrica: Métrica de distancia a utilizar. **Ventajas**: Ofrece una visión granular sobre la calidad de agrupamiento de cada muestra. **Limitaciones**: Las mismas que silhouette_score; sensible a los valores atípicos y a la métrica de distancia.	`from sklearn.metrics import silhouette_samples`
		X: Datos utilizados en agrupamiento
		etiquetas: Etiquetas de clúster para cada muestra
		`samples = silhouette_samples(X, labels, metric='euclidean')`

davies_bouldin_score	Mide la relación de similitud promedio de cada clúster con el clúster más similar. Valores más bajos indican un mejor agrupamiento. **Ventajas**: Proporciona una evaluación de agrupamiento simple y efectiva. **Limitaciones**: Puede no funcionar bien con clústeres altamente desbalanceados.	`from sklearn.metrics import davies_bouldin_score`
		X: Datos utilizados en clustering
		etiquetas: Etiquetas de clúster para cada muestra
		`db_score = davies_bouldin_score(X, labels)`

Voronoi	Calcula el diagrama de Voronoi, que partitiona el espacio basado en el vecino más cercano. **Pros**: Útil para análisis espacial y agrupamiento. **Limitaciones**: Limitado a casos de uso que implican la partición espacial de datos.	`from scipy.spatial import Voronoi`
		puntos: Coordenadas para el diagrama de Voronoi
		`vor = Voronoi(puntos)`

voronoi_plot_2d	Grafica el diagrama de Voronoi en 2D para visualizar los resultados de agrupamiento. **Hiperparámetros**: show_vertices: Si se deben mostrar los vértices. **Ventajas**: Ideal para visualizar agrupamientos espaciales. **Limitaciones**: Limitado a espacios en 2D y conjuntos de datos grandes pueden causar problemas de rendimiento.	`from scipy.spatial import voronoi_plot_2d`
		vor: Objeto de diagrama de Voronoi
		`voronoi_plot_2d(vor, show_vertices=True)`

matplotlib.patches.Patch	Crea formas personalizadas como rectángulos, círculos o elipses para añadir a gráficos. **Hiperparámetros**: color: Rellena el color de la forma. **Pros**: Versátil para la personalización visual. **Limitaciones**: Puede que no soporte todas las formas o personalizaciones complejas.	`import matplotlib.patches as patches`
		Crear un rectángulo con ancho, alto y posición espe
		`rectangle = patches.Rectangle((0, 0), 1, 1, color='blue')`

explained_variance_score	Mide la proporción de la varianza explicada por las predicciones del modelo. Un puntaje más alto indica un mejor rendimiento. **Pros**: Ayuda a evaluar el ajuste de los modelos de regresión. **Limitaciones**: No es adecuado para tareas de clasificación.	`from sklearn.metrics import explained_variance_score`
		y_true: Valores verdaderos
		y_pred: Valores predichos
		`ev_score = explained_variance_score(y_true, y_pred)`

Regresión Ridge	Realiza regresión ridge (regularización L2) para evitar el sobreajuste penalizando coeficientes grandes. **Hiperparámetros**: alpha: Fuerza de regularización. **Ventajas**: Ayuda a reducir el sobreajuste en modelos de regresión. **Limitaciones**: Puede no funcionar bien con datos dispersos.	`from sklearn.linear_model import Ridge`
		alpha: Fuerza de regularización (valores más grande
		`ridge = Ridge(alpha=1.0)`

Regresión Lasso	Realiza regresión lasso (regularización	`from sklearn.linear_model import Lasso`

	L1), que fomenta la escasez penalizando el valor absoluto de los coeficientes. **Hiperparámetros**: alpha: Fuerza de regularización. **Ventajas**: Fomenta soluciones escasas, útil para la selección de características. **Limitaciones**: Puede tener dificultades con la multicolinealidad.	**alpha: Fuerza de regularización (valores más grande** `lasso = Lasso(alpha=0.1)`

Pipeline	Encadena múltiples pasos de preprocesamiento y modelado en un solo objeto, asegurando un flujo de trabajo eficiente. **Pros**: Simplifica el código, asegura reproducibilidad. **Limitaciones**: Puede no funcionar bien con pipelines complejos que requieren configuraciones dinámicas.	`from sklearn.pipeline import Pipeline`
		pasos: Lista de tuplas con nombre y estimador/trans
		`pipeline = Pipeline(steps=[('escalador', StandardScaler()), ('modelo', Ridge(alpha=1.0))])`

GridSearchCV	Realiza una búsqueda exhaustiva sobre una cuadrícula de parámetros especificada para encontrar la mejor configuración del modelo. **Hiperparámetros**: param_grid: Diccionario de cuadrículas de parámetros. **Ventajas**: Asegura parámetros óptimos del modelo. **Limitaciones**: Costoso computacionalmente para cuadrículas grandes.	`from sklearn.model_selection import GridSearchCV`
		estimador: Modelo a ajustar
		param_grid: Diccionario con parámetros para buscar
		`grid_search = GridSearchCV(estimator=Ridge(), param_grid={'alpha': [0.1, 1.0, 10.0]})`
		Estrategias de visualización para la evaluación de k-means
		Nombre del Proceso
		Descripción Breve
		Fragmento de Código
		Múltiples ejecuciones de k-means
		Ejecuta el clustering KMeans múltiples veces con diferentes inicializaciones aleatorias para evaluar
		Ventaja: Ayuda a visualizar la consistencia.
		Limitación: Costoso computacionalmente para grandes conjuntos de datos.
		`# Número de ejecuciones para KMeans con diferentes estados aleatorios`
		`n_runs = 4`
		`inertia_values = []`
		`plt.figure(figsize=(12, 12))`

```
# Ejecutar K-Means múltiples veces con diferentes estados aleatorios
for i in range(n_runs):
    kmeans = KMeans(n_clusters=4, random_state=None)  # Usar el `n_init` por defecto
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

    # Graficar el resultado del clustering
    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
    plt.title(f'Ejecución de Clustering K-Means {i + 1}')
    plt.xlabel('Característica 1')
    plt.ylabel('Característica 2')
    plt.legend()

plt.tight_layout()
plt.show()

# Imprimir valores de inercia
for i, inertia in enumerate(inertia_values, start=1):
    print(f'Ejecución {i}: Inercia={inertia:.2f}')
```

Método del codo

Evalúa el número óptimo de clústeres graficando la inercia (suma de cuadrados dentro del clúster) pa

Ventaja: Fácil de interpretar.

Limitación: Punto de codo subjetivo.

```

# Rango de valores de k a probar
k_values = range(2, 11)

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

    # Calcular y almacenar métricas
    inertia_values.append(kmeans.inertia_)

# Graficar los valores de inercia (Método del codo)
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.plot(k_values, inertia_values, marker='o')
```

```
plt.title('Método del Codo: Inercia vs. k')
plt.xlabel('Número de Clústeres (k)')
plt.ylabel('Inercia')
```

Método de Silhouette

```
Determina el número óptimo de clústeres evaluando las puntuaciones de Silhouette para diferentes val
```

Ventaja: Considera tanto la cohesión como la separación.

Limitación: Alta computación para grandes conjuntos de datos.

```
# Rango de valores de k a probar
k_values = range(2, 11)

# Almacenar métricas de rendimiento
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))

# Graficar las puntuaciones de Silhouette
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 2)
plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Puntuación de Silhouette vs. k')
plt.xlabel('Número de Clústeres (k)')
plt.ylabel('Puntuación de Silhouette')
```

Índice de Davies-Bouldin

```
Evalúa el rendimiento del clustering calculando el DBI para diferentes valores de k.
```

Ventaja: Cuantifica la compacidad y la separación.

Limitación: Sensible a las formas y densidades de los clústeres.

```
# Rango de valores de k a probar
k_values = range(2, 11)

# Almacenar métricas de rendimiento
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))

# Graficar el 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 Clústeres (k)')
plt.ylabel('Índice de Davies-Bouldin')
```

Autores

Jeff Grossman
Abhishek Gagneja