MÉTRICAS DE CLASIFICACIÓN CON SCIKIT-LEARN

Abraham Jain Jiménez

Métricas de ajuste

Las métricas de ajuste se usan para monitorear y medir el rendimiento de un modelo de Machine Learning durante el entrenamiento y la prueba. Estas dependen de si el problema es de clasificación o de regresión.

En problemas reales no siempre se usan los modelos con las mejores métricas, pues las necesidades del negocio pueden jugar un papel importante en la elección del modelo. Por ejemplo, puede preferirse un modelo de clasificación más robusto, con mejor generalización ante nuevos datos que uno con clasificación perfecta pero muy sensible a ligeros cambios en los datos.

En este documento solo se abordarán las métricas para modelos de clasificación.

Métricas de clasificación

La matriz de confusión es una visualización tabular de las predicciones del modelo frente a las etiquetas reales. Su papel es evaluar el rendimiento del modelo mediante una comparación predicciones-valores reales.

Para clasificación binaria la matriz de confusión luce como:

	Predicho (negativo)	**Predicho (positivo)**
Real (negativo)	**TN**	**FP**
Real (positivo)	**FN**	**TP**

Table 1: Matriz de confusión binaria

Caso	Descripción	Ejemplo
True Positive	Se predijo positivo y es positivo	Predices que una mujer está embarazada y sí lo está
True Negative	Predices negativo y es negativo	Predices que una mujer no está embarazada y no lo está
False Positive	Predices positivo y es negativo	Predices que una mujer está embarazada pero no lo está
False Negative	Predices negativo y es positivo	Predices que una mujer no está embarazada pero sí lo está

Métricas derivadas de la matriz de confusión binaria

- **Accuracy:**

  Es la relación entre el número de predicciones correctas y el número total de muestras de entrada.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  Es una métrica útil en clases balanceadas, pero puede ser engañosa si hay desbalance. Idealmente, Accuracy ≈ 1 (100%).

- **Precision:**

  Indica qué tan confiable es el modelo cuando predice un caso como positivo.

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Es la relación positivos correctos-positivos predichos. Idealmente, Precision ≈ 1 (100%).

- **Recall:**

  Es la fracción de muestras de una clase que el modelo predice correctamente.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Se prioriza en situaciones donde es peor no detectar un True (por ejemplo, en tumores es mejor detectar un FP que un FN). En la literatura también se le suele identificar como *True Positive Rate (TPR)* o *Sensitivity*. Idealmente, Recall ≈ 1 (100%).

- **Specificity:**

  Es la fracción de muestras negativas predichas correctamente por el modelo.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

  En la literatura también se le suele identificar como *True Negative Rate (TNR)*. Idealmente, Specificity ≈ 1 (100%).

- **F1-Score:**

  Es la media armónica entre Precison y Recall. Intenta encontrar el equilibrio entre estas métricas. Es útil cuando ambas métricas son importantes.

$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

  Buena opción en desbalance de clases. Ofrece una mejor evaluación del modelo sin favorecer la Precision ni el Recall. Idealmente, $F1 \approx 1$ (100%).

- **False Positive Rate (FPR)**:

  Responde a la pregunta: De todos los casos FP, ¿cuántos predijo el modelo correctamente?

$$\text{FPR} = \frac{FP}{TN + FP}$$

- **False Negative Rate (FNR)**:

  Responde a la pregunta: De todos los casos FN, ¿cuántos predijo el modelo correctamente?

$$\text{Recall} = \frac{FN}{TP + FN}$$

Sintaxis de las métricas en `scikit-learn`

```
from sklearn.module import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size,
                                                    random_state)
model = Model()
model.fit(X_train, y_train)
y_pred=model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Modelos basados en puntaje

Dado un dato de entrada X, el modelo produce un valor estimador $\hat{p}(X) \in [0, 1]$, el cual es el grado de confianza del modelo de clasificación binaria de que X pertenece a la clase positiva.

El umbral $\tau \in [0, 1]$ convierte la probabilidad predicha $\hat{p}(X)$ en una predicción de clase:

$$\hat{y}(X) = \begin{cases} 1, & \text{si } \hat{p}(X) \geq \tau \\ 0, & \text{si } \hat{p}(X) < \tau \end{cases}$$

Métricas de resumen

- **ROC (*Receiver Operating Characteristic*) Curve:**
  Es la gráfica resultado de escanear el umbral en el intervalo $0 \leq \tau \leq 1$. Es decir, la curva ROC se construye variando τ, en donde en cada punto se grafica

$$\frac{TP}{TP + FN} = TPR(\tau) \text{ vs } FPR(\tau) = \frac{FP}{FP + TN}$$

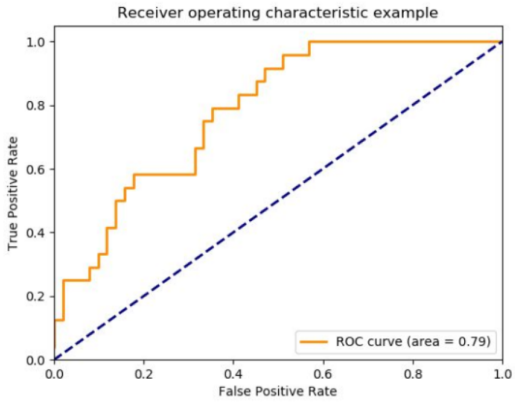

Figure 1: Curva ROC

Mientras más cercana a 1 sea el área bajo la curva ROC (AUC-ROC), es mejor el desempeño del modelo.

- **PRC (*Precision-Recall Curve*):**
  Muestra la relación

$$\frac{TP}{TP+FP} = \text{Precision}(\tau) \text{ vs Recall}(\tau) = \frac{TP}{TP+FN}$$

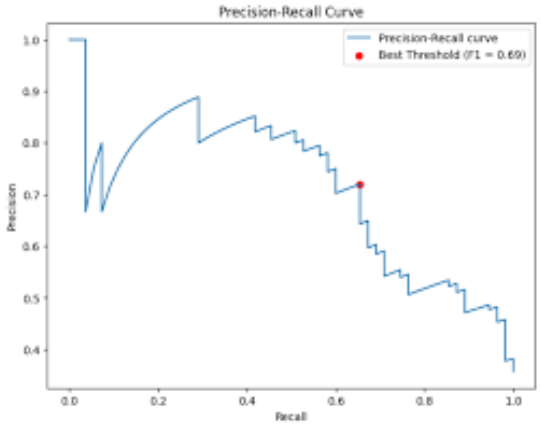

Figure 2: PRC

Mientras más cercana a 1 sea el área bajo la PRC (AUC-PRC), es mejor el desempeño del modelo.

Sobre la métrica AUC-ROC (*Area Under the Curve of ROC*)

Mide la capacidad del modelo para discriminar entre clases. Se calcula mediante:

$$AUC - ROC = \int_0^1 TPR(FPR)\, d(FPR)$$

Que por el método del trapecio se aproxima mediante:

$$AUC - ROC = \sum_{i=1}^{n-1} (FPR_{i+1} - FPR_i)\, \frac{TPR_{i+1} + TPR_i}{2}$$

- $AUC - ROC = 1.0$: El modelo de clasificación es perfecto.
- $AUC - ROC = 0.5$: Modelo sin capacidad de discriminación (aleatorio).
- $AUC - ROC < 0.5$: Modelo peor que el azar.

La métrica AUC-PRC es análoga en cuanto a fundamentos.

ROC vs PRC

Escenario	**Mejor Métrica**
Clases Balanceadas	ROC-AUC
Clases Desbalanceadas	PR-AUC

Sintaxis de AUC-ROC en `scikit-learn`

```
from sklearn.model_selection import train_test_split
from sklearn.module import Model
from sklearn.metrics import (roc_curve, auc, RocCurveDisplay)
import matplotlib.pyplot as plt


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size,
                                                    random_state)

model = Model()
```

```
model.fit(X_train, y_train)

#-Obtención  de las probabilidades para la clase positiva-
y_scores = model.predict_proba(X_test)[:, 1]

#-CURVA ROC y AUC-ROC-
fpr, tpr, thresholds_roc = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)
RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc).plot()
plt.title('Curva ROC')
plt.show()
```

Sintaxis de AUC-PRC en `scikit-learn`

```
from sklearn.model_selection import train_test_split
from sklearn.module import Model
from sklearn.metrics import (precision_recall_curve,
                             average_precision_score,
                             PrecisionRecallDisplay)
import matplotlib.pyplot as plt


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size,
                                                    random_state)

model = Model()
model.fit(X_train, y_train)

#-Obtención  de las probabilidades para la clase positiva-
y_scores = model.predict_proba(X_test)[:, 1]

# -PRC y AUC-PR-
precision, recall, thresholds_prc = precision_recall_curve(y_test, y_scores)
ap_score = average_precision_score(y_test, y_scores)
PrecisionRecallDisplay(precision=precision, recall=recall,
                                    average_precision=ap_score).plot()
```

```
plt.title('Curva Precision-Recall')
plt.show()
```

Consideraciones en `scikit-learn`

- Tanto la curva ROC como la curva PR están diseñadas para evaluar el modelo en función de su capacidad para detectar la clase positiva, por ello la nececidad de la línea de código:

  ```
  y_scores = model.predict_proba(X_test)[:, 1]
  ```

- Las sintaxis aplican con cualquier modelo compatible con el método `predict_proba()`. Siendo que ROC y PRC se basan en el umbral de decisión probabilístico τ, este método es el que permite obtener la probabilidad de pertenecer a cada clase.
- Comprobar si un modelo tiene el método `predict_proba()`:

  ```
  hasattr(model, "predict_proba")
  ```

  (Si tiene, devuelve un `True`)

Métricas para clasificación multiclase

Hasta ahora la teoría ha abarcado clasificación binaria, pero extender la lógica para multiclase es natural.

Si tenemos un conjunto de n clases, $C = \{C_1, \ldots, C_n\}$, la matriz de confusión se ve como:

Clase Real / Clase Predicha	C_1	$\cdots$	C_n
C_1	a_{11}	$\cdots$	a_{1n}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_n	a_{n1}	$\cdots$	a_{nn}

Las métricas AUC-ROC y AUC-PRC nacen originalmente para problemas binarios, pero se extienden para problemas multiclase mediante distintos enfoques, en donde los más comunes son OvR (*One-vs-Rest*) y OvO (*One-vs-One*).

One-vs-Rest (OvR):

- El problema se trabaja como un problema binario en donde se toma la clase C_i como positiva y todas las demás $n-1$ clases como negativas, luego se calcula AUC-ROC para cada clase individualmente.
- De lo anterior se tienen n curvas ROC y n valores de AUC-ROC.
- Se promedian todas las AUC-ROC para obtener un AUC global. Este promedio puede ser macro (promedio simple entre clases) o ponderado (promedio ponderado por el número de muestras por clase).
- La métrica AUC-PRC se trabaja de manera análoga.

One-vs-One (OvO):

- Se toma el conjuto de pares posibles de clases, $\{(i,j)\}$ y se calcula el AUC-ROC para cada uno de ellos. Es decir, se abordan múltiples problemas binarios.
- Lo anterior arroja $\frac{n(n-1)}{2}$ curvas ROC con sus respectivas AUC.
- Se promedian los resultados.
- La métrica AUC-PRC se trabaja de manera análoga.
- OvO es útil en desbalance de clases.

Las métricas derivadas de la matriz de confusión multiclase, Accuracy, Precision, Recall, Specificity, F1-Score, FPR y FNR, se calculan con base a los problemas binarios asociados a los enfoques OvR u OvO.

Sintaxis de AUC-ROC y AUC-PRC multiclase en `scikit-learn`

```
from sklearn.module import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

```

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size,
                                                    random_state)

model = Model()
model.fit(X_train, y_train)

# Predecir probabilidades de pertenencia a cada clase, {P(C_1), ... , P(C_n)}
y_proba = model.predict_proba(X_test)

# AUC-ROC con enfoque OvR
auc_ovo = roc_auc_score(y_test, y_proba, multi_class='ovr', average)
print("La métrica AUC-ROC por OvR es:", auc_ovo)

# AUC-PRC con enfoque OvR
auc_prc_ovo = average_precision_score(y_test, y_proba, average="macro")
print("La métrica AUC-PRC por OvR es:", auc_prc_ovo)

# AUC-ROC con enfoque OvO
auc_ovo = roc_auc_score(y_test, y_proba, multi_class="ovo", average="macro")
print("La métrica AUC-ROC por OvO es:", auc_ovo)

# Para AUC-PRC con enfoque OvO, scikit-learn no tiene un parámetro
```

Evaluando múltiples modelos de clasificación: Ejemplo

Digamos que queremos evaluar el rendimiento de tres modelos de clasificación que toman una base de datos en formato dataframe de Pandas. Los tres modelos son K-NN, Regresión Logística y un Árbol de Decisión.

El corazón de este ejemplo está en la métrica de evaluación. Por defecto en los modelos de clasificación de `scikit-learn` es Accuracy, pero dependiendo del problema y sus condiciones, como priorizar un FP sobre un FN o tener un desbalance de clases, puede usarse otra métrica.

```python
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

X = df.drop("columna", axis=1).values
y = df['columna'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

#Modelos de clasificación
models = {"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(), "Decision Tree"
results = []

#Resultados de la métrica Accuracy (la por default en cross_val_score)
for model in models.values():
     kf = KFold(n_splits=6, random_state=42, shuffle=True)
     cv_results = cross_val_score(model, X_train, y_train, cv=kf)
     results.append(cv_results)
print(f'Los seis valores de la métrica Accuracy para cada modelo son:\n {results}')

#Hallando el mejor modelo con respecto a Accuracy
for name, model in models.items():
  model.fit(X_train, y_train)
  test_score = model.score(X_test, y_test)
  print(f" Para el modelo {name}, la Accuracy en el test set es: {test_score}")
```