Hyperparameter Tuning

In python

Porqué es necesario optimizar un modelo

- Las combinaciones de parámetros en un modelo pueden ser en si mismo un problema NP
- Python ofrece herramientas para realizar experimentos con varios valores e incluso automatizar la salida de esos para devolver la mejor combinación
- Esa optimización también depende de los datos por lo que es común incluir dentro de la misma un sistema de validación (kfoldcross, etc…)

Métricas

- Para poder obtener una optimización de un modelo es necesario tener una medida de la calidad

Inglés	**Español**
Precision	Precisión
Recall	Exhaustividad
F1-score	Valor-F
Accuracy	Exactitud
Confusion Matrix	Matriz de Confusión
True Positive	Positivos Verdaderos
True Negative	Negativos Verdaderos
False Positive	Positivos Falsos
False Negative	Negativos Falsos

Ejemplo de marketing

Preguntas a 100 clientes y 80 de ellos nos dicen que no están interesados y 20 de ellos que sí.

True Negative [TN], True Positive [TP], False Positive [FP], False Negative [FN].

		predicción	
		0	**1**
realidad	**0**	70	10
	1	15	5

		predicción	
		0	**1**
realidad	**0**	TN	FP
	1	FN	TP

Matriz de Confusión con un ejemplo de marketing

0: no está interesado 1: sí está interesado

Accuracy (exactitud)

El accuracy (exactitud) se calcula con la siguiente fórmula:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

La exactitud del ejemplo de marketing sería:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{5 + 70}{5 + 70 + 10 + 15} = 0.75$$

Interpretación: el modelo acierta el 75% de las veces. La exactitud es una métrica muy engañosa. De hecho, si tuviésemos un modelo que siempre predijera que el cliente nunca va a estar interesado, su accuracy sería del 80%.

Peligro: la métrica accuracy (exactitud) no funciona bien cuando las clases están desbalanceadas

Precisión

Con la métrica de precisión podemos medir la **calidad** del modelo de machine learning en tareas de clasificación. En el ejemplo, se refiere a que la precisión es la respuesta a la pregunta ¿qué porcentaje de los clientes que contactemos estarán interesados?

Para calcular la precisión usaremos la siguiente fórmula:

$$precision = \frac{TP}{TP + FP} \qquad precision = \frac{TP}{TP + FP} = \frac{5}{5 + 10} = 0.33$$

		predicción	
		0	1
realidad	0	TN	FP
	1	FN	TP

sólo un 33% de los clientes a los que contactemos estarán realmente interesados. Esto significa que el modelo del ejemplo se equivocará un 66% de las veces cuando prediga que un cliente va a estar interesado.

Precisión (precision)

Recall (exhaustividad)

La métrica de exhaustividad nos va a informar sobre la **cantidad** que el modelo de machine learning es capaz de identificar. En el ejemplo, se refiere a que la exhaustividad (recall) es la respuesta a la pregunta ¿qué porcentaje de los clientes están interesados somos capaces de identificar?

Para calcular la exhaustividad (recall) usaremos la siguiente fórmula:

$$recall = \frac{TP}{TP + FN}$$

$$recall = \frac{TP}{TP + FN} = \frac{5}{5 + 15} = 0.25$$

		predicción	
		0	**1**
realidad	**0**	TN	FP
	1	FN	TP

el modelo sólo es capaz de identificar un 25% de los clientes que estarían interesados en adquirir el producto. Esto significa que el modelo del ejemplo sólo es capaz de identificar 1 de cada 4 de los clientes que sí aceptarían la oferta

F1

- El valor F1 se utiliza para combinar las medidas de precision y recall en un sólo valor. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones.

F1 se calcula haciendo la media armónica entre la precisión y la exhaustividad:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

En el ejemplo de marketing, combinando precision y recall en F1 nos quedaría:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.33 \cdot 0.25}{0.33 + 0.25} = 0.28$$

Modelos comunes de optimización

- Aleatoria: conjuntos de valores aleatorios para los parámetros indicados
- En grid (o rejilla) : conjutnos indicados, el modelo realizará el produco cartesiano de todos los valores indicados
- Siguiendo un método de optimización :
  - Los resultados parciales de ejecuciones anteriores sirven para guiar la búsqueda de parámetros en áreas prometoras
  - Algoritmos basados en metaheuristicas (genéticos, etc)

Ejemplo

Para una comprensión más clara, supongamos que queremos entrenar un clasificador de bosque aleatorio con el siguiente conjunto de hiperparámetros.

- n_estimadores: [100, 150, 200]
- profundidad_máxima: [20, 30, 40]

	n_estimators		
max_depth	(100, 20)	(150, 20)	(200, 20)
	(100, 30)	(150, 30)	(200, 30)
	(100, 40)	(150, 40)	(200, 40)

Ejemplo en python

```
from sklearn.model_selection import GridSearchCV
grid_vals = {'penalty': ['l1','l2'], 'C': [0.001,0.01,0.1,1]}
grid_lr = GridSearchCV(estimator=model, param_grid=grid_vals, scoring='accuracy’,  cv=6, refit=True, return_train_score=True)
grid_lr.fit(X_train, y_train)
preds = grid_lr.best_estimator_.predict(X_test)
```

Una vez entrenado, podemos acceder al mejor modelo, a la mejor puntuación y a los mejores parámetros encontrados a través de los atributos **best_estimator_**, **best_score_** y **best_params_** respectivamente

Grid search

- La búsqueda en cuadrícula es fácil de implementar para encontrar el mejor modelo dentro de la cuadrícula.
- Sin embargo, es computacionalmente costoso ya que el número del modelo continúa multiplicándose cuando agregamos nuevos valores de hiperparámetro
- Los valores son elegidos por un experto no habiendo garantía del resultado

Grid aleatorio

- Al igual que la búsqueda en cuadrícula, seguimos configurando los valores de hiperparámetros que queremos ajustar en Búsqueda aleatoria.
- Sin embargo, el modelo no entrena cada combinación de hiperparámetros, sino que los selecciona aleatoriamente.
- Tenemos que definir el número de muestras que queremos elegir de nuestra rejilla.

```
model = RandomForestClassifier()
param_vals = {'max_depth': [200, 500, 800, 1100], 'n_estimators': [100, 200, 300, 400],
'learning_rate': [0.001, 0.01, 0.1, 1, 10]]}
random_rf = RandomizedSearchCV(estimator=model, param_distributions=param_vals,
n_iter=10, scoring='accuracy', cv=5,
refit=True, n_jobs=-1)

#Training and prediction

random_rf.fit(X_train, y_train)
preds = random_rf.best_estimator_.predict(X_test)
```

Búsqueda informada

- A diferencia de la búsqueda en cuadrícula y aleatoria, la búsqueda informada aprende de sus iteraciones anteriores a través del siguiente proceso
  - Búsqueda aleatoria
  - Encuentra áreas con buena puntuación
  - Ejecute la búsqueda de cuadrícula en un área más pequeña
  - Continuar hasta obtener la solución óptima

Ejemplos en: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/ajuste-de-hiperparametros