Análisis Detallado: Optimización de Hiperparámetros

Bootcamp VirtIA

`r Sys.Date()`

Contents

Análisis Detallado: Optimización de Hiperparámetros

He analizado la celda "8. Optimización de Hiperparámetros" del notebook. Aquí está el análisis completo:

1. Razones del Tiempo de Ejecución Prolongado

Configuración Actual

```
# path=Telco_Customer_Churn.ipynb mode=EXCERPT
random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions,
    n_iter=50,
    cv=5,
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=1
)
```

Detalles técnicos:

- **Algoritmo:** `RandomizedSearchCV` (búsqueda aleatoria, más eficiente que GridSearchCV)
- **Modelo:** `RandomForestClassifier` (computacionalmente costoso)
- **Iteraciones:** 50 combinaciones aleatorias
- **Validación cruzada:** 5 folds (StratifiedKFold implícito)
- **Dataset:** 8,278 filas × 39 columnas (después de SMOTE)
- **Paralelización:** `n_jobs=-1` (ya activada)

Espacio de Búsqueda

```
# path=Telco_Customer_Churn.ipynb mode=EXCERPT
param_distributions = {
    'n_estimators': [100, 200, 300, 500],      # 4 opciones
    'max_depth': [10, 20, 30, None],           # 4 opciones
    'min_samples_split': [2, 5, 10],           # 3 opciones
    'min_samples_leaf': [1, 2, 4],             # 3 opciones
    'max_features': ['sqrt', 'log2'],          # 2 opciones
    'bootstrap': [True, False]                 # 2 opciones
}
```

Combinaciones posibles: $4 \times 4 \times 3 \times 3 \times 2 \times 2 = \mathbf{576}$ **combinaciones**
Combinaciones evaluadas: 50 (aleatorias)
Total de entrenamientos: $50 \times 5 = \mathbf{250}$ **fits** (confirmado en output)

---

2. Cálculo Estimado del Tiempo

Fórmula de Tiempo

```
Tiempo Total = (n_iter) × (cv_folds) × (tiempo_por_modelo)
```

Estimación Detallada

Factor	Valor	Impacto
Iteraciones	50	Alto
Folds CV	5	Alto
Total fits	250	Muy Alto
Muestras	8,278	Alto (duplicado por SMOTE)
Features	39	Moderado
Max n_estimators	500	Muy Alto

Complejidad de Random Forest:

```
O(n_estimators × n_samples × n_features × log(n_samples))
O(500 × 8,278 × 39 × log(8,278))
O(500 × 8,278 × 39 × 13.02)  2,100 millones de operaciones por modelo
```

Tiempo estimado por modelo:

- Con `n_estimators=100`: ~2-3 segundos
- Con `n_estimators=500`: ~10-15 segundos
- **Promedio:** ~5 segundos por fit

Tiempo total estimado:

```
250 fits × 5 segundos = 1,250 segundos  20-21 minutos
```

Por qué es tan costoso:

1. **Random Forest con 500 árboles** es extremadamente pesado
2. **Dataset balanceado con SMOTE** casi duplicó el tamaño (5,634 → 8,278 filas)
3. **5 folds de CV** multiplica el trabajo por 5
4. **50 iteraciones** es un número alto para búsqueda aleatoria
5. **39 features** después del preprocesamiento aumenta la complejidad

---

3. Recomendaciones para Reducir el Tiempo

Opción 1: Reducción Moderada (Recomendada)

Cambios sugeridos:

```python
random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions_reduced,
    n_iter=20,          #  Reducir de 50 a 20 (-60%)
    cv=3,               #  Reducir de 5 a 3 (-40%)
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=1
```

```
)

# Espacio de búsqueda reducido
param_distributions_reduced = {
    'n_estimators': [100, 200, 300],        #  Eliminar 500
    'max_depth': [10, 20, None],            #  Eliminar 30
    'min_samples_split': [2, 5],            #  Eliminar 10
    'min_samples_leaf': [1, 2],             #  Eliminar 4
    'max_features': ['sqrt', 'log2'],       #  Mantener
    'bootstrap': [True, False]              #  Mantener
}
```

Resultado:

- **Fits totales:** $20 \times 3 = \mathbf{60}$ **fits** (vs 250 original)
- **Tiempo estimado:** 60×3 seg = **180 segundos 3 minutos**
- **Reducción: 85% más rápido** (de ~20 min a ~3 min)
- **Pérdida de precisión:** Mínima (~0.5-1% en ROC-AUC)

Opción 2: Reducción Agresiva (Más Rápida)

```
# Muestreo estratificado del dataset
from sklearn.model_selection import train_test_split

X_train_sample, _, y_train_sample, _ = train_test_split(
    X_train_balanced,
    y_train_balanced,
    train_size=0.5,  # Usar solo 50% de los datos
    stratify=y_train_balanced,
    random_state=RANDOM_STATE
)

random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions_reduced,
    n_iter=15,          #  Reducir a 15
    cv=3,               #  3 folds
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=1
)

random_search.fit(X_train_sample, y_train_sample)  #  Usar muestra
```

Resultado:

- **Dataset:** 4,139 muestras (50% del balanceado)

- **Fits totales:** $15 \times 3 = 45$ **fits**
- **Tiempo estimado:** 45×1.5 seg = **67 segundos 1 minuto**
- **Reducción: 95% más rápido**
- **Pérdida de precisión:** Moderada (~1-2% en ROC-AUC)

Opción 3: Valores Acotados por Conocimiento Previo

Basándome en los resultados obtenidos, puedo acotar aún más:

```
# path=Telco_Customer_Churn.ipynb mode=EXCERPT
# Mejores hiperparámetros encontrados:
{'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 1,
 'max_features': 'log2', 'max_depth': 20, 'bootstrap': False}
```

Espacio acotado basado en resultados:

```
param_distributions_focused = {
    'n_estimators': [200, 300, 400],        # Rango cercano a 500
    'max_depth': [15, 20, 25],              # Rango cercano a 20
    'min_samples_split': [2, 3],            # Valores bajos
    'min_samples_leaf': [1, 2],             # Valores bajos
    'max_features': ['log2'],               # Solo el mejor
    'bootstrap': [False, True]              # Priorizar False
}
```

Resultado:

- **Combinaciones posibles:** $3 \times 3 \times 2 \times 2 \times 1 \times 2 = 72$
- **Con n_iter=15, cv=3:** $15 \times 3 = 45$ **fits**
- **Tiempo: ~2 minutos**
- **Precisión:** Muy similar al original

---

4. Alternativas Más Rápidas

Estrategia Híbrida (Mejor Opción)

```
# FASE 1: Búsqueda rápida con espacio amplio
param_distributions_phase1 = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False]
}
```

```
random_search_phase1 = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions_phase1,
    n_iter=10,  # Solo 10 iteraciones
    cv=3,
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1
)

random_search_phase1.fit(X_train_balanced, y_train_balanced)
best_params_phase1 = random_search_phase1.best_params_

# FASE 2: Refinamiento alrededor de los mejores parámetros
param_distributions_phase2 = {
    'n_estimators': [best_params_phase1['n_estimators'],
                     best_params_phase1['n_estimators'] + 100,
                     best_params_phase1['n_estimators'] + 200],
    'max_depth': [best_params_phase1['max_depth']],  # Fijo
    'min_samples_split': [best_params_phase1['min_samples_split']],  # Fijo
    'min_samples_leaf': [best_params_phase1['min_samples_leaf']],  # Fijo
    'max_features': [best_params_phase1['max_features']],  # Fijo
    'bootstrap': [best_params_phase1['bootstrap'],
                  not best_params_phase1['bootstrap']]
}

random_search_phase2 = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions_phase2,
    n_iter=6,
    cv=5,  # Más riguroso en fase 2
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1
)

random_search_phase2.fit(X_train_balanced, y_train_balanced)
```

Ventajas:

- **Tiempo total:** ~5-7 minutos (vs 20 minutos)
- **Precisión:** Similar o mejor que búsqueda única
- **Exploración + Explotación:** Mejor balance

Comparación de Opciones

Opción	Tiempo	Fits	Precisión	Recomendación
Original	~20 min	250	100%	Muy lento

Opción	Tiempo	Fits	Precisión	Recomendación
Moderada	~3 min	60	~99%	**Mejor balance**
Agresiva	~1 min	45	~98%	Solo para pruebas rápidas
Acotada	~2 min	45	~99.5%	Si conoces buenos rangos
Híbrida	~5-7 min	60	~99.5%	Mejor exploración

Recomendación Final para Google Colab

Para ejecutar en Google Colab de forma eficiente, usa la **Opción 1 (Moderada)**:

```
# Configuración optimizada para Colab
param_distributions_colab = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False]
}

random_search_colab = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions_colab,
    n_iter=20,
    cv=3,
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=2  # Más información durante ejecución
)

random_search_colab.fit(X_train_balanced, y_train_balanced)
```

Beneficios:

- **Tiempo:** ~3 minutos (85% más rápido)
- **Precisión:** Pérdida mínima (<1%)
- **Recursos:** Compatible con Colab gratuito
- **Iteraciones:** Permite múltiples experimentos