In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_auc_score,
    roc_curve, precision_recall_curve, average_precision_score
)
import pickle
import os
import warnings
warnings.filterwarnings('ignore')

# ============================================================================
# CONFIGURACIÓN
# ============================================================================
TEST_SIZE = 0.2
RANDOM_STATE = 42
CV_FOLDS = 5
N_ESTIMATORS = 100
MAX_DEPTH = 15
MIN_SAMPLES_SPLIT = 10
MIN_SAMPLES_LEAF = 5

# Directorio de salida
OUTPUT_DIR = "../../data/random-forest-output"
os.makedirs(OUTPUT_DIR, exist_ok=True)

print("="*80)
print("MODELO DE RANDOM FOREST - HOME CREDIT DEFAULT RISK")
print("="*80)
print(f"\nTipo de problema: CLASIFICACIÓN BINARIA (0 = Pago, 1 = Default)")
print(f"Criterio de split: GINI (mide impureza para clasificación)")
print(f"Directorio de salida: {OUTPUT_DIR}")
```

```
============================================================================
====
MODELO DE RANDOM FOREST - HOME CREDIT DEFAULT RISK
============================================================================
====

Tipo de problema: CLASIFICACIÓN BINARIA (0 = Pago, 1 = Default)
Criterio de split: GINI (mide impureza para clasificación)
Directorio de salida: ../../data/random-forest-output
```

In [2]:
```python
# ============================================================================
# 1. CARGA Y VERIFICACIÓN DE DATOS
# ============================================================================
print("\n[1/15] Cargando y verificando datos...")

df = pd.read_csv("../../data/processed/variables.csv")
```

```python
print(f"Dataset cargado: {df.shape}")
print(f"Usuarios: {len(df):,}")
print(f"Variables: {len(df.columns)}")
print(f"Tasa de default: {df['TARGET'].mean()*100:.2f}%")
```

```
[1/15] Cargando y verificando datos...
Dataset cargado: (307511, 41)
Usuarios: 307,511
Variables: 41
Tasa de default: 8.07%
```

In [3]:

```python
# ===========================================================================
# 2. PREPARACIÓN DE DATOS
# ===========================================================================
print("\n[2/15] Preparando datos...")

X = df.drop(['SK_ID_CURR', 'TARGET'], axis=1)
y = df['TARGET']

print(f"Variables predictoras: {X.shape[1]}")
print(f"Variable objetivo: {y.name}")

numeric_vars = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_vars = X.select_dtypes(include=['object']).columns.tolist()

print(f"Variables numericas: {len(numeric_vars)}")
print(f"Variables categoricas: {len(categorical_vars)}")
```

```
[2/15] Preparando datos...
Variables predictoras: 39
Variable objetivo: TARGET
Variables numericas: 34
Variables categoricas: 5
```

In [ ]:

```python
# ===========================================================================
# 3. MANEJO DE VALORES FALTANTES
# ===========================================================================
print("\n[3/15] Analizando valores faltantes...")

missing_pct = (X.isnull().sum() / len(X) * 100).sort_values(ascending=False)
missing_vars = missing_pct[missing_pct > 0]

if len(missing_vars) > 0:
    print(f"Variables con valores faltantes: {len(missing_vars)}")
    print(missing_vars.head(10))

    print("Imputando valores faltantes...")
    for col in numeric_vars:
        if X[col].isnull().sum() > 0:
            X[col].fillna(X[col].median(), inplace=True)

    for col in categorical_vars:
        if X[col].isnull().sum() > 0:
            X[col].fillna(X[col].mode()[0], inplace=True)

    print("Valores faltantes imputados")
```

```python
else:
    print("No hay valores faltantes")
```

```
[3/15] Analizando valores faltantes...
Variables con valores faltantes: 10
TASA_INTERES_PROMEDIO    98.501192
RATIO_PAGO_MINIMO_TC     80.726218
EXT_SOURCE_1             56.381073
EXT_SOURCE_3             19.825307
PLAZO_PROMEDIO            5.485657
RATIO_PAGO_CUOTA          5.163718
EXT_SOURCE_2              0.214626
SCORE_PROMEDIO            0.055933
AMT_ANNUITY               0.003902
CNT_FAM_MEMBERS           0.000650
dtype: float64
Imputando valores faltantes...
Valores faltantes imputados
```

In [5]:

```python
# ==============================================================================
# 4. CODIFICACIÓN DE VARIABLES CATEGÓRICAS
# ==============================================================================
print("\n[4/15] Codificando variables categoricas...")

label_encoders = {}

if len(categorical_vars) > 0:
    for col in categorical_vars:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))
        label_encoders[col] = le

    print(f"{len(categorical_vars)} variables codificadas")
    print(f"Categorias: {categorical_vars}")
```

```
[4/15] Codificando variables categoricas...
5 variables codificadas
Categorias: ['NAME_FAMILY_STATUS', 'CODE_GENDER', 'NAME_EDUCATION_TYPE', 'FL
AG_OWN_CAR', 'FLAG_OWN_REALTY']
```

In [6]:
```
# ==========================================================================
# 5. DIVISIÓN TRAIN/TEST
# ==========================================================================
print("\n[5/15] Dividiendo datos en Train/Test...")

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

print(f"Train set: {X_train.shape[0]:,} usuarios ({X_train.shape[0]/len(X)*1
print(f"Test set:  {X_test.shape[0]:,} usuarios ({X_test.shape[0]/len(X)*100
print(f"\nDistribucion TARGET en Train:")
print(f"  Clase 0 (Pago): {(y_train==0).sum():,} ({(y_train==0).sum()/len(y_
print(f"  Clase 1 (Default): {(y_train==1).sum():,} ({(y_train==1).sum()/ler
```

```
[5/15] Dividiendo datos en Train/Test...
Train set: 246,008 usuarios (80.0%)
Test set:  61,503 usuarios (20.0%)

Distribucion TARGET en Train:
  Clase 0 (Pago): 226,148 (91.93%)
  Clase 1 (Default): 19,860 (8.07%)
```

In [7]:
```
# ==========================================================================
# 6. NOTA SOBRE NORMALIZACIÓN
# ==========================================================================
print("\n[6/15] Nota sobre normalizacion...")

# Random Forest NO requiere normalización ya que es un modelo basado en árbo
# Los árboles de decisión son invariantes a transformaciones monótonas de la
# Usamos los datos sin escalar para mejor interpretabilidad

print("Random Forest no requiere normalizacion de datos")
print("Los arboles de decision son invariantes a escalas de las variables")
print("Se usaran los datos originales (sin escalar)")
```

```
[6/15] Nota sobre normalizacion...
Random Forest no requiere normalizacion de datos
Los arboles de decision son invariantes a escalas de las variables
Se usaran los datos originales (sin escalar)
```

Random Forest 1

In [8]:
```
# ==========================================================================
# 7. ENTRENAMIENTO DEL MODELO
# ==========================================================================
print("\n[7/15] Entrenando Random Forest...")

model = RandomForestClassifier(
    n_estimators=N_ESTIMATORS,
    max_depth=MAX_DEPTH,
```

```python
    min_samples_split=MIN_SAMPLES_SPLIT,
    min_samples_leaf=MIN_SAMPLES_LEAF,
    class_weight='balanced',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=1
)

model.fit(X_train, y_train)

print("\nModelo entrenado exitosamente")
print(f"Numero de arboles: {model.n_estimators}")
print(f"Profundidad maxima: {model.max_depth}")
print(f"Min samples split: {model.min_samples_split}")
print(f"Min samples leaf: {model.min_samples_leaf}")
print(f"Clases: {model.classes_}")
```

```
[7/15] Entrenando Random Forest...
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.7s
Modelo entrenado exitosamente
Numero de arboles: 100
Profundidad maxima: 15
Min samples split: 10
Min samples leaf: 5
Clases: [0 1]
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   12.4s finished
```

In [9]:

```python
# ==============================================================================
# 8. VALIDACIÓN CRUZADA
# ==============================================================================
print(f"\n[8/15] Validacion Cruzada ({CV_FOLDS}-fold en Train)...")

cv_scores = cross_val_score(
    model, X_train, y_train,
    cv=CV_FOLDS,
    scoring='roc_auc',
    n_jobs=-1
)

print(f"ROC-AUC por fold: {cv_scores}")
print(f"Media: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")
```

```
[8/15] Validacion Cruzada (5-fold en Train)...
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   40.5s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   40.7s finished
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   40.7s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   40.8s finished
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.2s finished
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   41.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.3s finished
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.2s finished
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
ROC-AUC por fold: [0.74124295 0.7320359  0.74449844 0.7443346  0.74670445]
Media: 0.7418 (+/- 0.0052)
```

In [10]:

```
# ==============================================================================
# 9. PREDICCIONES
# ==============================================================================
print("\n[9/15] Generando predicciones...")

y_train_pred = model.predict(X_train)
y_train_proba = model.predict_proba(X_train)[:, 1]

y_test_pred = model.predict(X_test)
y_test_proba = model.predict_proba(X_test)[:, 1]

print("Predicciones generadas")
```

```
[9/15] Generando predicciones...
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.4s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.4s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
```

```
Predicciones generadas
```

```
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
```

In [11]:

```python
# ============================================================================
# 10. EVALUACIÓN EN TRAIN
# ============================================================================
print("\n[10/15] Evaluando modelo en TRAIN SET...")
print("="*80)

train_auc = roc_auc_score(y_train, y_train_proba)
train_ap = average_precision_score(y_train, y_train_proba)

print(f"\nMetricas Globales:")
print(f"ROC-AUC Score: {train_auc:.4f}")
print(f"Average Precision: {train_ap:.4f}")

print(f"\nClassification Report:")
print(classification_report(y_train, y_train_pred,
                            target_names=['Pago (0)', 'Default (1)']))

print(f"\nConfusion Matrix:")
cm_train = confusion_matrix(y_train, y_train_pred)
print(cm_train)
print(f"\nInterpretacion:")
print(f"  Verdaderos Negativos (TN): {cm_train[0,0]:,} - Predijo 'Pago' y si
print(f"  Falsos Positivos (FP): {cm_train[0,1]:,} - Predijo 'Default' pero
print(f"  Falsos Negativos (FN): {cm_train[1,0]:,} - Predijo 'Pago' pero hiz
print(f"  Verdaderos Positivos (TP): {cm_train[1,1]:,} - Predijo 'Default' y
```

```
[10/15] Evaluando modelo en TRAIN SET...
============================================================================
====

Metricas Globales:
ROC-AUC Score: 0.9222
Average Precision: 0.5242

Classification Report:
              precision    recall  f1-score   support

    Pago (0)       0.98      0.87      0.92    226148
 Default (1)       0.36      0.78      0.49     19860

    accuracy                           0.87    246008
   macro avg       0.67      0.83      0.71    246008
weighted avg       0.93      0.87      0.89    246008


Confusion Matrix:
[[197868  28280]
 [  4282  15578]]

Interpretacion:
  Verdaderos Negativos (TN): 197,868 - Predijo 'Pago' y si pago
  Falsos Positivos (FP): 28,280 - Predijo 'Default' pero pago
  Falsos Negativos (FN): 4,282 - Predijo 'Pago' pero hizo default [CRITICO]
  Verdaderos Positivos (TP): 15,578 - Predijo 'Default' y si hizo default
```

In [12]:

```python
# ============================================================================
# 11. EVALUACIÓN EN TEST
# ============================================================================
print("\n[11/15] Evaluando modelo en TEST SET (metricas definitivas)...")
print("="*80)

test_auc = roc_auc_score(y_test, y_test_proba)
test_ap = average_precision_score(y_test, y_test_proba)

print(f"\nMetricas Globales:")
print(f"ROC-AUC Score: {test_auc:.4f}")
print(f"Average Precision: {test_ap:.4f}")

print(f"\nClassification Report:")
print(classification_report(y_test, y_test_pred,
                            target_names=['Pago (0)', 'Default (1)']))

print(f"\nConfusion Matrix:")
cm_test = confusion_matrix(y_test, y_test_pred)
print(cm_test)
print(f"\nInterpretacion:")
print(f"  Verdaderos Negativos (TN): {cm_test[0,0]:,} - Predijo 'Pago' y si
print(f"  Falsos Positivos (FP): {cm_test[0,1]:,} - Predijo 'Default' pero p
print(f"  Falsos Negativos (FN): {cm_test[1,0]:,} - Predijo 'Pago' pero hizc
print(f"  Verdaderos Positivos (TP): {cm_test[1,1]:,} - Predijo 'Default' y
```

```
[11/15] Evaluando modelo en TEST SET (metricas definitivas)...
================================================================================
====

Metricas Globales:
ROC-AUC Score: 0.7451
Average Precision: 0.2207

Classification Report:
              precision    recall  f1-score   support

    Pago (0)       0.95      0.86      0.90     56538
 Default (1)       0.21      0.44      0.29      4965

    accuracy                           0.82     61503
   macro avg       0.58      0.65      0.59     61503
weighted avg       0.89      0.82      0.85     61503


Confusion Matrix:
[[48497  8041]
 [ 2777  2188]]

Interpretacion:
  Verdaderos Negativos (TN): 48,497 - Predijo 'Pago' y si pago
  Falsos Positivos (FP): 8,041 - Predijo 'Default' pero pago
  Falsos Negativos (FN): 2,777 - Predijo 'Pago' pero hizo default [CRITICO]
  Verdaderos Positivos (TP): 2,188 - Predijo 'Default' y si hizo default
```

In [13]:

```python
# ============================================================================
# 12. COMPARACIÓN TRAIN VS TEST
# ============================================================================
print("\n[12/15] Comparando TRAIN vs TEST (analisis de overfitting)...")
print("="*80)

print(f"\n{'Metrica':<25} {'Train':<12} {'Test':<12} {'Diferencia':<12}")
print("-" * 65)
print(f"{'ROC-AUC':<25} {train_auc:<12.4f} {test_auc:<12.4f} {abs(train_auc-
print(f"{'Average Precision':<25} {train_ap:<12.4f} {test_ap:<12.4f} {abs(tr

diff_auc = abs(train_auc - test_auc)
if diff_auc < 0.02:
    print("\nExcelente generalizacion (diferencia < 2%)")
elif diff_auc < 0.05:
    print("\nBuena generalizacion (diferencia < 5%)")
elif diff_auc < 0.10:
    print("\nPosible ligero overfitting (diferencia 5-10%)")
else:
    print("\nOverfitting detectado (diferencia > 10%)")
```

```
[12/15] Comparando TRAIN vs TEST (analisis de overfitting)...
================================================================================

Metrica                   Train        Test         Diferencia
-----------------------------------------------------------------
ROC-AUC                   0.9222       0.7451       0.1771
Average Precision         0.5242       0.2207       0.3035

Overfitting detectado (diferencia > 10%)
```

In [14]:

```python
# ==============================================================================
# 13. IMPORTANCIA DE VARIABLES
# ==============================================================================
print("\n[13/15] Calculando importancia de variables...")
print("="*80)

# Random Forest usa importancia basada en Gini (mean decrease in impurity)
# GINI se usa porque es un problema de CLASIFICACIÓN (predecir 0 o 1)
# - Gini mide la impureza de los nodos: que tan mezcladas están las clases
# - Un nodo "puro" tiene solo una clase (Gini = 0)
# - Un nodo con 50/50 tiene máxima impureza (Gini = 0.5 para binario)

feature_importance = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': model.feature_importances_
}).sort_values('Importancia', ascending=False)

print("\nTop 20 variables mas importantes para predecir default:\n")
print(feature_importance.head(20).to_string(index=False))

print("\nInterpretacion de importancia (Gini Importance):")
print("  - Problema de CLASIFICACIÓN BINARIA: predecir 0 (Pago) o 1 (Default
print("  - Gini mide cuanto reduce cada variable la impureza en los splits")
print("  - Valores mas altos = mayor capacidad para separar las clases")
print("  - Basado en reduccion promedio de impureza en los 100 arboles")
```

```
[13/15] Calculando importancia de variables...
================================================================================
====

Top 20 variables mas importantes para predecir default:

               Variable  Importancia
          SCORE_PROMEDIO     0.183249
            EXT_SOURCE_2     0.095282
            EXT_SOURCE_3     0.094086
              DAYS_BIRTH     0.042976
               EDAD_ANOS     0.042558
            EXT_SOURCE_1     0.037583
        RATIO_PAGO_CUOTA     0.036051
             AMT_ANNUITY     0.035872
              AMT_CREDIT     0.034971
     CREDIT_INCOME_RATIO     0.033973
          PLAZO_PROMEDIO     0.031336
  TOTAL_CREDITO_OTORGADO     0.030732
 TOTAL_CREDITO_HISTORICO     0.030321
   MONTO_PROMEDIO_PREVIO     0.030230
      TOTAL_DEUDA_ACTUAL     0.024201
      INGRESO_PER_CAPITA     0.023310
        AMT_INCOME_TOTAL     0.021557
       CREDITOS_CERRADOS     0.016803
    RATIO_PAGO_MINIMO_TC     0.016604
   NUM_PRESTAMOS_PREVIOS     0.016398

Interpretacion de importancia (Gini Importance):
  - Problema de CLASIFICACIÓN BINARIA: predecir 0 (Pago) o 1 (Default)
  - Gini mide cuanto reduce cada variable la impureza en los splits
  - Valores mas altos = mayor capacidad para separar las clases
  - Basado en reduccion promedio de impureza en los 100 arboles
```

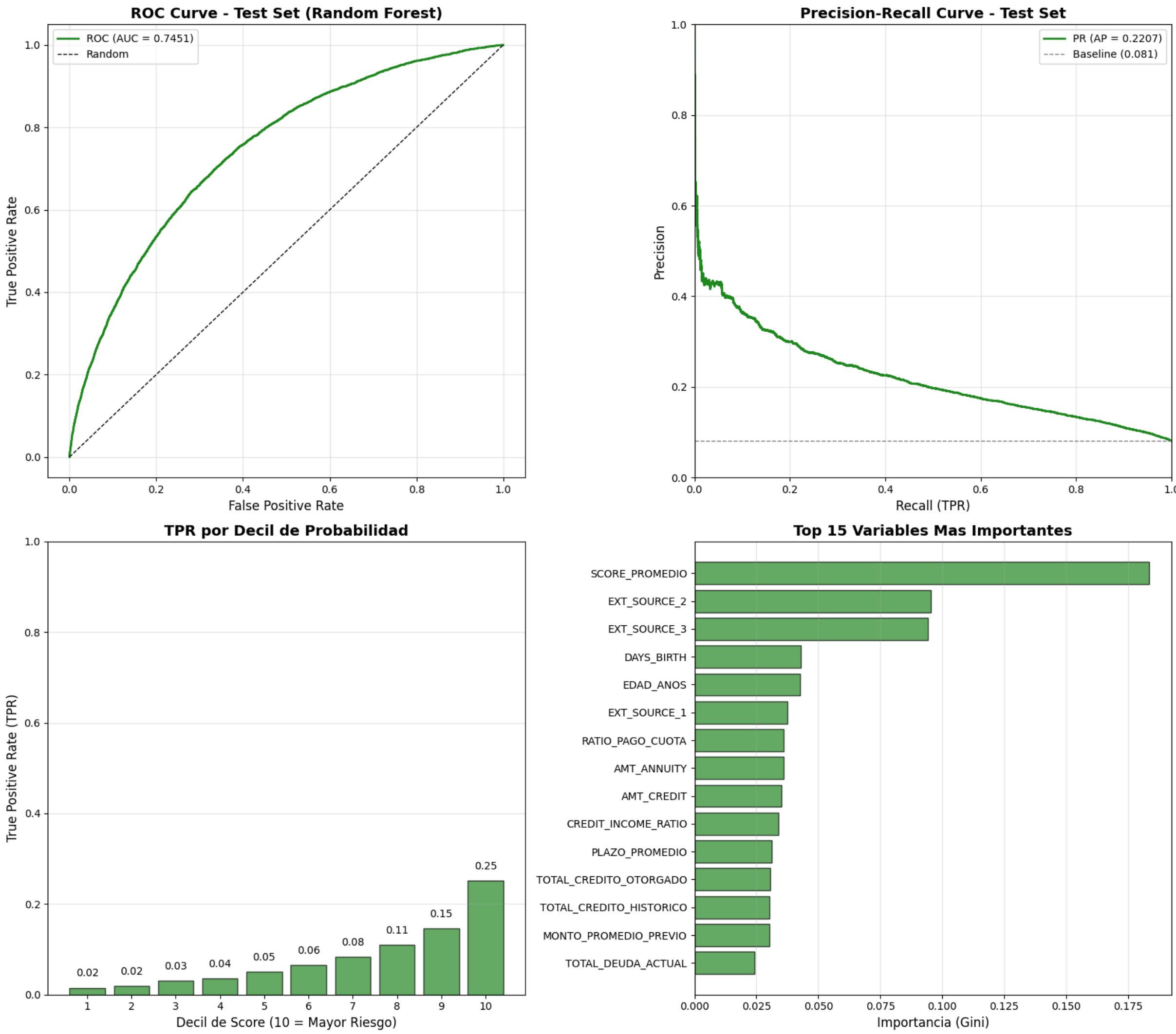
In [15]:

```python
# ==============================================================================
# 14. VISUALIZACIONES
# ==============================================================================
print("\n[14/15] Generando visualizaciones...")

# Calcular deciles
test_results = pd.DataFrame({
    'y_true': y_test,
    'y_proba': y_test_proba
})
test_results['decil'] = pd.qcut(test_results['y_proba'], q=10, labels=False,

# Calcular TPR por decil
decil_stats = test_results.groupby('decil').agg({
    'y_true': ['sum', 'count', 'mean']
}).reset_index()
decil_stats.columns = ['decil', 'positivos', 'total', 'tpr']
decil_stats = decil_stats.sort_values('decil', ascending=False)

# Crear gráficas
fig, axes = plt.subplots(2, 2, figsize=(16, 14))
```

```python
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
axes[0, 0].plot(fpr, tpr, linewidth=2, color='forestgreen', label=f'ROC (AUC
axes[0, 0].plot([0, 1], [0, 1], 'k--', linewidth=1, label='Random')
axes[0, 0].set_xlabel('False Positive Rate', fontsize=12)
axes[0, 0].set_ylabel('True Positive Rate', fontsize=12)
axes[0, 0].set_title('ROC Curve - Test Set (Random Forest)', fontsize=14, fo
axes[0, 0].legend()
axes[0, 0].grid(alpha=0.3)

# Precision-Recall Curve
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_test_pro
sort_idx = np.argsort(recall)
recall_sorted = recall[sort_idx]
precision_sorted = precision[sort_idx]

axes[0, 1].plot(recall_sorted, precision_sorted, linewidth=2, color='forestg
axes[0, 1].axhline(y=y_test.mean(), color='gray', linestyle='--', linewidth=
axes[0, 1].set_xlabel('Recall (TPR)', fontsize=12)
axes[0, 1].set_ylabel('Precision', fontsize=12)
axes[0, 1].set_title('Precision-Recall Curve - Test Set', fontsize=14, fontw
axes[0, 1].set_xlim([0, 1])
axes[0, 1].set_ylim([0, 1])
axes[0, 1].legend()
axes[0, 1].grid(alpha=0.3)

# TPR por Decil
axes[1, 0].bar(decil_stats['decil'], decil_stats['tpr'], color='forestgreen'
axes[1, 0].set_xlabel('Decil de Score (10 = Mayor Riesgo)', fontsize=12)
axes[1, 0].set_ylabel('True Positive Rate (TPR)', fontsize=12)
axes[1, 0].set_title('TPR por Decil de Probabilidad', fontsize=14, fontweigh
axes[1, 0].set_xticks(range(1, 11))
axes[1, 0].grid(alpha=0.3, axis='y')
axes[1, 0].set_ylim(0, 1)

for i, row in decil_stats.iterrows():
    axes[1, 0].text(row['decil'], row['tpr'] + 0.02, f"{row['tpr']:.2f}",
                ha='center', va='bottom', fontsize=10)

# Feature Importance (Top 15)
top_features = feature_importance.head(15)
axes[1, 1].barh(range(len(top_features)), top_features['Importancia'].values
axes[1, 1].set_yticks(range(len(top_features)))
axes[1, 1].set_yticklabels(top_features['Variable'].values)
axes[1, 1].set_xlabel('Importancia (Gini)', fontsize=12)
axes[1, 1].set_title('Top 15 Variables Mas Importantes', fontsize=14, fontwe
axes[1, 1].invert_yaxis()
axes[1, 1].grid(alpha=0.3, axis='x')

plt.tight_layout()
plt.savefig(f'{OUTPUT_DIR}/random_forest_curves.png', dpi=300, bbox_inches='
print(f"Graficas guardadas: {OUTPUT_DIR}/random_forest_curves.png")

plt.show()

# Imprimir tabla de deciles
```

```python
print("\nTabla de TPR por Decil:")
print(decil_stats.to_string(index=False))
```

```
[14/15] Generando visualizaciones...
Graficas guardadas: ../../data/random-forest-output/random_forest_curves.png
```




```
Tabla de TPR por Decil:
 decil  positivos  total       tpr
    10       1546   6151  0.251341
     9        899   6150  0.146179
     8        679   6150  0.110407
     7        510   6150  0.082927
     6        399   6150  0.064878
     5        309   6151  0.050236
     4        219   6150  0.035610
     3        191   6150  0.031057
     2        120   6150  0.019512
     1         93   6151  0.015119
```

In [16]:
```python
# ==========================================================================
# 15. GUARDAR MODELO Y RESULTADOS
# ==========================================================================
print("\n[15/15] Guardando modelo y resultados...")

with open(f'{OUTPUT_DIR}/random_forest_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

```python
with open(f'{OUTPUT_DIR}/label_encoders_rf.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)

feature_importance.to_csv(f'{OUTPUT_DIR}/feature_importance_rf.csv', index=F

test_predictions = pd.DataFrame({
    'SK_ID_CURR': df.iloc[X_test.index]['SK_ID_CURR'].values,
    'TARGET_Real': y_test.values,
    'TARGET_Predicho': y_test_pred,
    'Probabilidad_Default': y_test_proba
})
test_predictions.to_csv(f'{OUTPUT_DIR}/test_predictions_rf.csv', index=False

print(f"Archivos guardados en: {OUTPUT_DIR}/")
print("  - random_forest_model.pkl")
print("  - label_encoders_rf.pkl")
print("  - feature_importance_rf.csv")
print("  - test_predictions_rf.csv")
print("  - random_forest_curves.png")

# ===============================================================================
# RESUMEN FINAL
# ===============================================================================
print("\n" + "="*80)
print("ANALISIS COMPLETADO - RANDOM FOREST")
print("="*80)
print(f"\nTipo de problema: CLASIFICACIÓN BINARIA")
print(f"  - Objetivo: Predecir si un cliente hará default (1) o pagará (0)")
print(f"  - Criterio de split: GINI (mide impureza para clasificación)")
print(f"\nResultados Finales (TEST SET):")
print(f"  ROC-AUC: {test_auc:.4f}")
print(f"  Average Precision: {test_ap:.4f}")
print(f"  Accuracy: {(y_test_pred == y_test).mean():.4f}")
print(f"\nConfiguracion del modelo:")
print(f"  N_ESTIMATORS: {N_ESTIMATORS}")
print(f"  MAX_DEPTH: {MAX_DEPTH}")
print(f"  MIN_SAMPLES_SPLIT: {MIN_SAMPLES_SPLIT}")
print(f"  MIN_SAMPLES_LEAF: {MIN_SAMPLES_LEAF}")
print(f"\nArchivos guardados en: {OUTPUT_DIR}/")
print(f"\nEl modelo esta listo para produccion")
print("="*80)
```

```
[15/15] Guardando modelo y resultados...
Archivos guardados en: ../../data/random-forest-output/
  - random_forest_model.pkl
  - label_encoders_rf.pkl
  - feature_importance_rf.csv
  - test_predictions_rf.csv
  - random_forest_curves.png

============================================================================
====
ANALISIS COMPLETADO - RANDOM FOREST
============================================================================
====

Tipo de problema: CLASIFICACIÓN BINARIA
  - Objetivo: Predecir si un cliente hará default (1) o pagará (0)
  - Criterio de split: GINI (mide impureza para clasificación)

Resultados Finales (TEST SET):
  ROC-AUC: 0.7451
  Average Precision: 0.2207
  Accuracy: 0.8241

Configuracion del modelo:
  N_ESTIMATORS: 100
  MAX_DEPTH: 15
  MIN_SAMPLES_SPLIT: 10
  MIN_SAMPLES_LEAF: 5

Archivos guardados en: ../../data/random-forest-output/

El modelo esta listo para produccion
============================================================================
====
```

Random forest 2

In [17]:
```python
# Drop de variables redundantes/derivadas
df = df.drop(columns=[
    # Edad
    'DAYS_BIRTH',

    # Montos base
    'AMT_CREDIT',
    'AMT_INCOME_TOTAL',
    'AMT_ANNUITY',

    # Scores individuales
    'EXT_SOURCE_1',
    'EXT_SOURCE_2',
    'EXT_SOURCE_3',

    # Familia
    'CNT_FAM_MEMBERS',
```

```
    # Activos
    'FLAG_OWN_CAR',
    'FLAG_OWN_REALTY',

    # Consultas buró total
    'TOTAL_CONSULTAS_BURO',

    # Componentes de ratios (solo MESES_CON_MORA está en df)
    'MESES_CON_MORA',

    # Variables binarias derivadas
    'TIENE_IMPAGOS',
    'ES_PRIMER_CREDITO',

    # Créditos buró
    'CANTIDAD_CREDITOS_BURO'
])
```

In [18]:
```
# =========================================================================
# RANDOM FOREST 2 - MODELO CON VARIABLES REDUCIDAS
# =========================================================================
print("="*80)
print("RANDOM FOREST 2 - MODELO CON VARIABLES REDUCIDAS")
print("="*80)

print(f"\nDataset despues de drop: {df.shape}")
print(f"Variables eliminadas: 14")
print(f"Variables restantes: {len(df.columns)}")
```

```
============================================================================
====
RANDOM FOREST 2 - MODELO CON VARIABLES REDUCIDAS
============================================================================
====

Dataset despues de drop: (307511, 26)
Variables eliminadas: 14
Variables restantes: 26
```

In [19]:
```
# =========================================================================
# PREPARACIÓN DE DATOS - MODELO 2
# =========================================================================
print("\n[2/15] Preparando datos para modelo 2...")

X2 = df.drop(['SK_ID_CURR', 'TARGET'], axis=1)
y2 = df['TARGET']

print(f"Variables predictoras: {X2.shape[1]}")
print(f"Variable objetivo: {y2.name}")

numeric_vars2 = X2.select_dtypes(include=[np.number]).columns.tolist()
categorical_vars2 = X2.select_dtypes(include=['object']).columns.tolist()

print(f"Variables numericas: {len(numeric_vars2)}")
print(f"Variables categoricas: {len(categorical_vars2)}")
```

```python
print(f"\nVariables en el modelo:")
for i, col in enumerate(X2.columns, 1):
    print(f"  {i}. {col}")
```

```
[2/15] Preparando datos para modelo 2...
Variables predictoras: 24
Variable objetivo: TARGET
Variables numericas: 21
Variables categoricas: 3

Variables en el modelo:
  1. EDAD_ANOS
  2. SCORE_PROMEDIO
  3. CREDIT_INCOME_RATIO
  4. NAME_FAMILY_STATUS
  5. CNT_CHILDREN
  6. CODE_GENDER
  7. NAME_EDUCATION_TYPE
  8. INGRESO_PER_CAPITA
  9. NUM_ACTIVOS
  10. TOTAL_CREDITO_DISPONIBLE
  11. TOTAL_CREDITO_OTORGADO
  12. TOTAL_DEUDA_ACTUAL
  13. MAX_DIAS_MORA
  14. CREDITOS_ACTIVOS
  15. CREDITOS_CERRADOS
  16. PCT_MESES_MORA
  17. CREDITOS_CON_IMPAGO
  18. NUM_PRESTAMOS_PREVIOS
  19. TASA_INTERES_PROMEDIO
  20. PLAZO_PROMEDIO
  21. MONTO_PROMEDIO_PREVIO
  22. TOTAL_CREDITO_HISTORICO
  23. RATIO_PAGO_CUOTA
  24. RATIO_PAGO_MINIMO_TC
```

In [20]:

```python
# ==============================================================================
# MANEJO DE VALORES FALTANTES - MODELO 2
# ==============================================================================
print("\n[3/15] Analizando valores faltantes...")

missing_pct2 = (X2.isnull().sum() / len(X2) * 100).sort_values(ascending=Fal
missing_vars2 = missing_pct2[missing_pct2 > 0]

if len(missing_vars2) > 0:
    print(f"Variables con valores faltantes: {len(missing_vars2)}")
    print(missing_vars2.head(10))

    print("\nImputando valores faltantes...")
    for col in numeric_vars2:
        if X2[col].isnull().sum() > 0:
            X2[col].fillna(X2[col].median(), inplace=True)

    for col in categorical_vars2:
        if X2[col].isnull().sum() > 0:
            X2[col].fillna(X2[col].mode()[0], inplace=True)
```

```
    print("Valores faltantes imputados")
else:
    print("No hay valores faltantes")
```

```
[3/15] Analizando valores faltantes...
Variables con valores faltantes: 5
TASA_INTERES_PROMEDIO    98.501192
RATIO_PAGO_MINIMO_TC     80.726218
PLAZO_PROMEDIO            5.485657
RATIO_PAGO_CUOTA          5.163718
SCORE_PROMEDIO            0.055933
dtype: float64

Imputando valores faltantes...
Valores faltantes imputados
```

In [21]:
```
# ==============================================================================
# CODIFICACIÓN DE VARIABLES CATEGÓRICAS - MODELO 2
# ==============================================================================
print("\n[4/15] Codificando variables categoricas...")

label_encoders2 = {}

if len(categorical_vars2) > 0:
    for col in categorical_vars2:
        le = LabelEncoder()
        X2[col] = le.fit_transform(X2[col].astype(str))
        label_encoders2[col] = le

    print(f"{len(categorical_vars2)} variables codificadas")
    print(f"Categorias: {categorical_vars2}")
else:
    print("No hay variables categoricas para codificar")
```

```
[4/15] Codificando variables categoricas...
3 variables codificadas
Categorias: ['NAME_FAMILY_STATUS', 'CODE_GENDER', 'NAME_EDUCATION_TYPE']
```

In [22]:
```
# ==============================================================================
# DIVISIÓN TRAIN/TEST - MODELO 2
# ==============================================================================
print("\n[5/15] Dividiendo datos en Train/Test...")

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE,
    stratify=y2
)

print(f"Train set: {X2_train.shape[0]:,} usuarios ({X2_train.shape[0]/len(X2
print(f"Test set:  {X2_test.shape[0]:,} usuarios ({X2_test.shape[0]/len(X2)*
print(f"\nDistribucion TARGET en Train:")
print(f"  Clase 0 (Pago): {(y2_train==0).sum():,} ({(y2_train==0).sum()/len(
print(f"  Clase 1 (Default): {(y2_train==1).sum():,} ({(y2_train==1).sum()/l
```

```
[5/15] Dividiendo datos en Train/Test...
Train set: 246,008 usuarios (80.0%)
Test set:  61,503 usuarios (20.0%)

Distribucion TARGET en Train:
  Clase 0 (Pago): 226,148 (91.93%)
  Clase 1 (Default): 19,860 (8.07%)
```

In [23]:
```
# ============================================================================
# ENTRENAMIENTO DEL MODELO 2
# ============================================================================
print("\n[7/15] Entrenando Random Forest 2 (variables reducidas)...")

model2 = RandomForestClassifier(
    n_estimators=N_ESTIMATORS,
    max_depth=MAX_DEPTH,
    min_samples_split=MIN_SAMPLES_SPLIT,
    min_samples_leaf=MIN_SAMPLES_LEAF,
    class_weight='balanced',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=1
)

model2.fit(X2_train, y2_train)

print("\nModelo 2 entrenado exitosamente")
print(f"Numero de arboles: {model2.n_estimators}")
print(f"Profundidad maxima: {model2.max_depth}")
print(f"Min samples split: {model2.min_samples_split}")
print(f"Min samples leaf: {model2.min_samples_leaf}")
print(f"Clases: {model2.classes_}")
```

```
[7/15] Entrenando Random Forest 2 (variables reducidas)...
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.0s
Modelo 2 entrenado exitosamente
Numero de arboles: 100
Profundidad maxima: 15
Min samples split: 10
Min samples leaf: 5
Clases: [0 1]
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   10.0s finished
```

In [24]:
```
# ============================================================================
# VALIDACIÓN CRUZADA - MODELO 2
# ============================================================================
print(f"\n[8/15] Validacion Cruzada ({CV_FOLDS}-fold en Train)...")

cv_scores2 = cross_val_score(
    model2, X2_train, y2_train,
    cv=CV_FOLDS,
    scoring='roc_auc',
    n_jobs=-1
)
```

```python
print(f"ROC-AUC por fold: {cv_scores2}")
print(f"Media: {cv_scores2.mean():.4f} (+/- {cv_scores2.std():.4f})")
```

```
[8/15] Validacion Cruzada (5-fold en Train)...
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.5s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.6s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.8s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.9s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.9s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   30.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   30.1s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   30.3s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   30.4s finished
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   30.4s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.2s finished
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.2s finished
ROC-AUC por fold: [0.73840893 0.725804   0.73686891 0.73917831 0.74439579]
Media: 0.7369 (+/- 0.0061)
```

In [25]:

```python
# ==========================================================================
# PREDICCIONES - MODELO 2
# ==========================================================================
print("\n[9/15] Generando predicciones...")

y2_train_pred = model2.predict(X2_train)
y2_train_proba = model2.predict_proba(X2_train)[:, 1]
```

```python
y2_test_pred = model2.predict(X2_test)
y2_test_proba = model2.predict_proba(X2_test)[:, 1]

print("Predicciones generadas")
```

```
[9/15] Generando predicciones...
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.4s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
```

```
Predicciones generadas
```

```
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.4s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
```

In [26]:

```python
# ==============================================================================
# EVALUACIÓN EN TRAIN - MODELO 2
# ==============================================================================
print("\n[10/15] Evaluando modelo 2 en TRAIN SET...")
print("="*80)

train2_auc = roc_auc_score(y2_train, y2_train_proba)
train2_ap = average_precision_score(y2_train, y2_train_proba)

print(f"\nMetricas Globales:")
print(f"ROC-AUC Score: {train2_auc:.4f}")
print(f"Average Precision: {train2_ap:.4f}")

print(f"\nClassification Report:")
print(classification_report(y2_train, y2_train_pred,
                            target_names=['Pago (0)', 'Default (1)']))

print(f"\nConfusion Matrix:")
cm2_train = confusion_matrix(y2_train, y2_train_pred)
print(cm2_train)
print(f"\nInterpretacion:")
print(f"  Verdaderos Negativos (TN): {cm2_train[0,0]:,} - Predijo 'Pago' y s
print(f"  Falsos Positivos (FP): {cm2_train[0,1]:,} - Predijo 'Default' perc
print(f"  Falsos Negativos (FN): {cm2_train[1,0]:,} - Predijo 'Pago' pero hi
print(f"  Verdaderos Positivos (TP): {cm2_train[1,1]:,} - Predijo 'Default'
```

```
[10/15] Evaluando modelo 2 en TRAIN SET...
============================================================================
====

Metricas Globales:
ROC-AUC Score: 0.9038
Average Precision: 0.5131

Classification Report:
              precision    recall  f1-score   support

    Pago (0)       0.98      0.86      0.91    226148
 Default (1)       0.32      0.76      0.45     19860

    accuracy                           0.85    246008
   macro avg       0.65      0.81      0.68    246008
weighted avg       0.92      0.85      0.88    246008


Confusion Matrix:
[[193755  32393]
 [  4744  15116]]

Interpretacion:
  Verdaderos Negativos (TN): 193,755 - Predijo 'Pago' y si pago
  Falsos Positivos (FP): 32,393 - Predijo 'Default' pero pago
  Falsos Negativos (FN): 4,744 - Predijo 'Pago' pero hizo default [CRITICO]
  Verdaderos Positivos (TP): 15,116 - Predijo 'Default' y si hizo default
```

In [27]:

```python
# ==============================================================================
# EVALUACIÓN EN TEST - MODELO 2
# ==============================================================================
print("\n[11/15] Evaluando modelo 2 en TEST SET (metricas definitivas)...")
print("="*80)

test2_auc = roc_auc_score(y2_test, y2_test_proba)
test2_ap = average_precision_score(y2_test, y2_test_proba)

print(f"\nMetricas Globales:")
print(f"ROC-AUC Score: {test2_auc:.4f}")
print(f"Average Precision: {test2_ap:.4f}")

print(f"\nClassification Report:")
print(classification_report(y2_test, y2_test_pred,
                            target_names=['Pago (0)', 'Default (1)']))

print(f"\nConfusion Matrix:")
cm2_test = confusion_matrix(y2_test, y2_test_pred)
print(cm2_test)
print(f"\nInterpretacion:")
print(f"  Verdaderos Negativos (TN): {cm2_test[0,0]:,} - Predijo 'Pago' y si
print(f"  Falsos Positivos (FP): {cm2_test[0,1]:,} - Predijo 'Default' pero
print(f"  Falsos Negativos (FN): {cm2_test[1,0]:,} - Predijo 'Pago' pero hiz
print(f"  Verdaderos Positivos (TP): {cm2_test[1,1]:,} - Predijo 'Default' y
```

```
[11/15] Evaluando modelo 2 en TEST SET (metricas definitivas)...
============================================================================
====

Metricas Globales:
ROC-AUC Score: 0.7407
Average Precision: 0.2133

Classification Report:
              precision    recall  f1-score   support

    Pago (0)       0.95      0.84      0.89     56538
 Default (1)       0.20      0.46      0.28      4965

    accuracy                           0.81     61503
   macro avg       0.58      0.65      0.59     61503
weighted avg       0.89      0.81      0.84     61503


Confusion Matrix:
[[47558  8980]
 [ 2661  2304]]

Interpretacion:
  Verdaderos Negativos (TN): 47,558 - Predijo 'Pago' y si pago
  Falsos Positivos (FP): 8,980 - Predijo 'Default' pero pago
  Falsos Negativos (FN): 2,661 - Predijo 'Pago' pero hizo default [CRITICO]
  Verdaderos Positivos (TP): 2,304 - Predijo 'Default' y si hizo default
```

In [28]:
```python
# ===========================================================================
# COMPARACIÓN TRAIN VS TEST - MODELO 2
# ===========================================================================
print("\n[12/15] Comparando TRAIN vs TEST (analisis de overfitting)...")
print("="*80)

print(f"\n{'Metrica':<25} {'Train':<12} {'Test':<12} {'Diferencia':<12}")
print("-" * 65)
print(f"{'ROC-AUC':<25} {train2_auc:<12.4f} {test2_auc:<12.4f} {abs(train2_a
print(f"{'Average Precision':<25} {train2_ap:<12.4f} {test2_ap:<12.4f} {abs(

diff_auc2 = abs(train2_auc - test2_auc)
if diff_auc2 < 0.02:
    print("\nExcelente generalizacion (diferencia < 2%)")
elif diff_auc2 < 0.05:
    print("\nBuena generalizacion (diferencia < 5%)")
elif diff_auc2 < 0.10:
    print("\nPosible ligero overfitting (diferencia 5-10%)")
else:
    print("\nOverfitting detectado (diferencia > 10%)")
```

```
[12/15] Comparando TRAIN vs TEST (analisis de overfitting)...
============================================================================
====

Metrica                    Train        Test         Diferencia
-----------------------------------------------------------------
ROC-AUC                    0.9038       0.7407       0.1631
Average Precision          0.5131       0.2133       0.2998

Overfitting detectado (diferencia > 10%)
```

In [29]:

```python
# ============================================================================
# IMPORTANCIA DE VARIABLES - MODELO 2
# ============================================================================
print("\n[13/15] Calculando importancia de variables...")
print("="*80)

feature_importance2 = pd.DataFrame({
    'Variable': X2.columns,
    'Importancia': model2.feature_importances_
}).sort_values('Importancia', ascending=False)

print("\nTop 20 variables mas importantes para predecir default:\n")
print(feature_importance2.head(20).to_string(index=False))

print("\nInterpretacion de importancia (Gini Importance):")
print("  - Problema de CLASIFICACIÓN BINARIA: predecir 0 (Pago) o 1 (Default
print("  - Gini mide cuanto reduce cada variable la impureza en los splits")
print("  - Valores mas altos = mayor capacidad para separar las clases")
print("  - Basado en reduccion promedio de impureza en los 100 arboles")
```

```
[13/15] Calculando importancia de variables...
============================================================================
====

Top 20 variables mas importantes para predecir default:

                 Variable  Importancia
            SCORE_PROMEDIO     0.321122
                 EDAD_ANOS     0.082152
       CREDIT_INCOME_RATIO     0.054617
          RATIO_PAGO_CUOTA     0.053444
    TOTAL_CREDITO_OTORGADO     0.050914
   TOTAL_CREDITO_HISTORICO     0.047529
      MONTO_PROMEDIO_PREVIO    0.046952
            PLAZO_PROMEDIO     0.045953
         TOTAL_DEUDA_ACTUAL    0.040421
         INGRESO_PER_CAPITA    0.038446
          CREDITOS_CERRADOS    0.030508
       RATIO_PAGO_MINIMO_TC    0.026664
        NAME_EDUCATION_TYPE    0.025050
      NUM_PRESTAMOS_PREVIOS    0.024877
           CREDITOS_ACTIVOS    0.023181
  TOTAL_CREDITO_DISPONIBLE     0.019034
               CODE_GENDER     0.017551
         NAME_FAMILY_STATUS    0.012794
             PCT_MESES_MORA    0.012441
               NUM_ACTIVOS     0.010104

Interpretacion de importancia (Gini Importance):
  - Problema de CLASIFICACIÓN BINARIA: predecir 0 (Pago) o 1 (Default)
  - Gini mide cuanto reduce cada variable la impureza en los splits
  - Valores mas altos = mayor capacidad para separar las clases
  - Basado en reduccion promedio de impureza en los 100 arboles
```

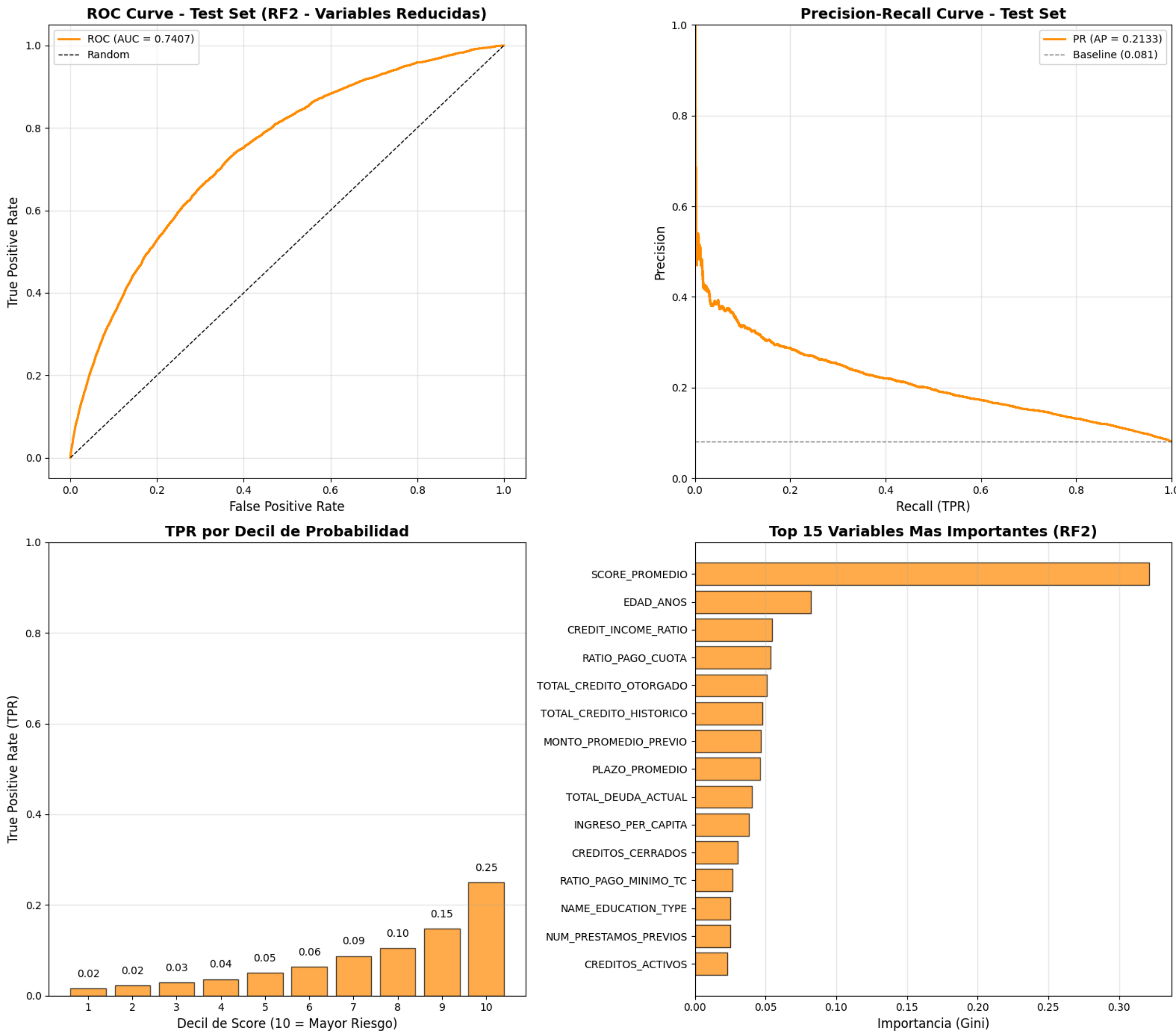
In [30]:

```
# ============================================================================
# VISUALIZACIONES - MODELO 2
# ============================================================================
print("\n[14/15] Generando visualizaciones...")

# Calcular deciles
test_results2 = pd.DataFrame({
    'y_true': y2_test,
    'y_proba': y2_test_proba
})
test_results2['decil'] = pd.qcut(test_results2['y_proba'], q=10, labels=Fals

# Calcular TPR por decil
decil_stats2 = test_results2.groupby('decil').agg({
    'y_true': ['sum', 'count', 'mean']
}).reset_index()
decil_stats2.columns = ['decil', 'positivos', 'total', 'tpr']
decil_stats2 = decil_stats2.sort_values('decil', ascending=False)

# Crear gráficas
fig, axes = plt.subplots(2, 2, figsize=(16, 14))
```

```python
# ROC Curve
fpr2, tpr2, _ = roc_curve(y2_test, y2_test_proba)
axes[0, 0].plot(fpr2, tpr2, linewidth=2, color='darkorange', label=f'ROC (AU
axes[0, 0].plot([0, 1], [0, 1], 'k--', linewidth=1, label='Random')
axes[0, 0].set_xlabel('False Positive Rate', fontsize=12)
axes[0, 0].set_ylabel('True Positive Rate', fontsize=12)
axes[0, 0].set_title('ROC Curve - Test Set (RF2 - Variables Reducidas)', for
axes[0, 0].legend()
axes[0, 0].grid(alpha=0.3)

# Precision-Recall Curve
precision2, recall2, _ = precision_recall_curve(y2_test, y2_test_proba)
sort_idx2 = np.argsort(recall2)
recall_sorted2 = recall2[sort_idx2]
precision_sorted2 = precision2[sort_idx2]

axes[0, 1].plot(recall_sorted2, precision_sorted2, linewidth=2, color='darko
axes[0, 1].axhline(y=y2_test.mean(), color='gray', linestyle='--', linewidth
axes[0, 1].set_xlabel('Recall (TPR)', fontsize=12)
axes[0, 1].set_ylabel('Precision', fontsize=12)
axes[0, 1].set_title('Precision-Recall Curve - Test Set', fontsize=14, fontw
axes[0, 1].set_xlim([0, 1])
axes[0, 1].set_ylim([0, 1])
axes[0, 1].legend()
axes[0, 1].grid(alpha=0.3)

# TPR por Decil
axes[1, 0].bar(decil_stats2['decil'], decil_stats2['tpr'], color='darkorange
axes[1, 0].set_xlabel('Decil de Score (10 = Mayor Riesgo)', fontsize=12)
axes[1, 0].set_ylabel('True Positive Rate (TPR)', fontsize=12)
axes[1, 0].set_title('TPR por Decil de Probabilidad', fontsize=14, fontweigh
axes[1, 0].set_xticks(range(1, 11))
axes[1, 0].grid(alpha=0.3, axis='y')
axes[1, 0].set_ylim(0, 1)

for i, row in decil_stats2.iterrows():
    axes[1, 0].text(row['decil'], row['tpr'] + 0.02, f"{row['tpr']:.2f}",
                ha='center', va='bottom', fontsize=10)

# Feature Importance (Top 15)
top_features2 = feature_importance2.head(15)
axes[1, 1].barh(range(len(top_features2)), top_features2['Importancia'].valu
axes[1, 1].set_yticks(range(len(top_features2)))
axes[1, 1].set_yticklabels(top_features2['Variable'].values)
axes[1, 1].set_xlabel('Importancia (Gini)', fontsize=12)
axes[1, 1].set_title('Top 15 Variables Mas Importantes (RF2)', fontsize=14,
axes[1, 1].invert_yaxis()
axes[1, 1].grid(alpha=0.3, axis='x')

plt.tight_layout()
plt.savefig(f'{OUTPUT_DIR}/random_forest2_curves.png', dpi=300, bbox_inches=
print(f"Graficas guardadas: {OUTPUT_DIR}/random_forest2_curves.png")

plt.show()

# Imprimir tabla de deciles
```

```python
print("\nTabla de TPR por Decil:")
print(decil_stats2.to_string(index=False))
```

```
[14/15] Generando visualizaciones...
Graficas guardadas: ../../data/random-forest-output/random_forest2_curves.pn
g
```




```
Tabla de TPR por Decil:
 decil  positivos  total      tpr
    10       1533   6151 0.249228
     9        912   6150 0.148293
     8        644   6150 0.104715
     7        538   6150 0.087480
     6        391   6150 0.063577
     5        313   6151 0.050886
     4        224   6150 0.036423
     3        178   6150 0.028943
     2        138   6150 0.022439
     1         94   6151 0.015282
```

In [31]:

```python
# ==========================================================================
# GUARDAR MODELO Y RESULTADOS - MODELO 2
# ==========================================================================
print("\n[15/15] Guardando modelo 2 y resultados...")

with open(f'{OUTPUT_DIR}/random_forest2_model.pkl', 'wb') as f:
```

```
    pickle.dump(model2, f)

with open(f'{OUTPUT_DIR}/label_encoders_rf2.pkl', 'wb') as f:
    pickle.dump(label_encoders2, f)

feature_importance2.to_csv(f'{OUTPUT_DIR}/feature_importance_rf2.csv', index

test_predictions2 = pd.DataFrame({
    'SK_ID_CURR': df.iloc[X2_test.index]['SK_ID_CURR'].values,
    'TARGET_Real': y2_test.values,
    'TARGET_Predicho': y2_test_pred,
    'Probabilidad_Default': y2_test_proba
})
test_predictions2.to_csv(f'{OUTPUT_DIR}/test_predictions_rf2.csv', index=Fal

print(f"Archivos guardados en: {OUTPUT_DIR}/")
print("  - random_forest2_model.pkl")
print("  - label_encoders_rf2.pkl")
print("  - feature_importance_rf2.csv")
print("  - test_predictions_rf2.csv")
print("  - random_forest2_curves.png")
```

```
[15/15] Guardando modelo 2 y resultados...
Archivos guardados en: ../../data/random-forest-output/
  - random_forest2_model.pkl
  - label_encoders_rf2.pkl
  - feature_importance_rf2.csv
  - test_predictions_rf2.csv
  - random_forest2_curves.png
```

In [ ]:

```
# ===========================================================================
# COMPARACIÓN MODELO 1 VS MODELO 2
# ===========================================================================
print("\n" + "="*80)
print("COMPARACIÓN: MODELO 1 (39 vars) VS MODELO 2 (25 vars)")
print("="*80)

print(f"\n{'Metrica':<25} {'RF1 (39 vars)':<15} {'RF2 (25 vars)':<15} {'Dife
print("-" * 70)
print(f"{'ROC-AUC (Test)':<25} {test_auc:<15.4f} {test2_auc:<15.4f} {test2_a
print(f"{'Average Precision (Test)':<25} {test_ap:<15.4f} {test2_ap:<15.4f}
print(f"{'ROC-AUC (Train)':<25} {train_auc:<15.4f} {train2_auc:<15.4f} {trai
print(f"{'Overfitting (Train-Test)':<25} {train_auc - test_auc:<15.4f} {trai

print(f"\nNumero de variables:")
print(f"  Modelo 1: {X.shape[1]} variables")
print(f"  Modelo 2: {X2.shape[1]} variables")
print(f"  Reduccion: {X.shape[1] - X2.shape[1]} variables eliminadas ({(X.sh

# Conclusión
if test2_auc >= test_auc:
    print(f"\n✓ Modelo 2 tiene MEJOR o IGUAL rendimiento con menos variables
    print(f"  Esto sugiere que las variables eliminadas eran redundantes")
else:
    print(f"\n✗ Modelo 2 tiene PEOR rendimiento")
    print(f"  Diferencia de AUC: {test_auc - test2_auc:.4f}")
```

```
print("\n" + "="*80)
print("ANALISIS COMPLETADO - RANDOM FOREST 2 (VARIABLES REDUCIDAS)")
print("="*80)
```

```
================================================================================
COMPARACIÓN: MODELO 1 (39 vars) VS MODELO 2 (25 vars)
================================================================================

Metrica                    RF1 (39 vars)    RF2 (25 vars)    Diferencia
----------------------------------------------------------------------
ROC-AUC (Test)             0.7451           0.7407           -0.0044
Average Precision (Test)   0.2207           0.2133           -0.0074
ROC-AUC (Train)            0.9222           0.9038           -0.0184
Overfitting (Train-Test)   0.1771           0.1631

Numero de variables:
  Modelo 1: 39 variables
  Modelo 2: 24 variables
  Reduccion: 15 variables eliminadas (38.5%)

✗ Modelo 2 tiene PEOR rendimiento
  Diferencia de AUC: 0.0044

================================================================================
ANALISIS COMPLETADO - RANDOM FOREST 2 (VARIABLES REDUCIDAS)
================================================================================
```

Optimización de hiperparametros

In [33]:

```
# ================================================================================
# RANDOM FOREST 3 - OPTIMIZACIÓN DE HIPERPARÁMETROS (MODELO 2)
# ================================================================================
# Usaremos RandomizedSearchCV para encontrar los mejores hiperparámetros
# para el modelo con variables reducidas (24 variables)

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

print("="*80)
print("RANDOM FOREST 3 - BÚSQUEDA DE HIPERPARÁMETROS ÓPTIMOS")
print("="*80)
print(f"\nUsando el dataset de {X2.shape[1]} variables (Modelo 2)")
print(f"Datos de entrenamiento: {X2_train.shape[0]:,} usuarios")
print(f"Datos de prueba: {X2_test.shape[0]:,} usuarios")
```

```
================================================================================
====
RANDOM FOREST 3 - BÚSQUEDA DE HIPERPARÁMETROS ÓPTIMOS
================================================================================
====

Usando el dataset de 24 variables (Modelo 2)
Datos de entrenamiento: 246,008 usuarios
Datos de prueba: 61,503 usuarios
```

In [34]:

```python
# ============================================================================
# DEFINICIÓN DEL ESPACIO DE BÚSQUEDA
# ============================================================================
print("\n[1/5] Definiendo espacio de búsqueda de hiperparámetros...")

# Espacio de hiperparámetros a explorar
param_distributions = {
    # Número de árboles - más árboles = mejor generalización pero más tiempo
    'n_estimators': [50, 100, 150, 200, 300],

    # Profundidad máxima - controla overfitting
    # Valores más bajos = menos overfitting
    'max_depth': [5, 8, 10, 12, 15, 20, None],

    # Mínimo de muestras para split - valores más altos = menos overfitting
    'min_samples_split': [2, 5, 10, 15, 20, 30, 50],

    # Mínimo de muestras en hoja - valores más altos = menos overfitting
    'min_samples_leaf': [1, 2, 5, 10, 15, 20, 30],

    # Número de features a considerar en cada split
    'max_features': ['sqrt', 'log2', 0.3, 0.5, 0.7],

    # Bootstrap samples
    'bootstrap': [True, False],

    # Criterio de split
    'criterion': ['gini', 'entropy']
}

# Calcular número total de combinaciones
total_combinations = 1
for key, values in param_distributions.items():
    total_combinations *= len(values)

print(f"\nHiperparámetros a optimizar:")
for param, values in param_distributions.items():
    print(f"  - {param}: {values}")

print(f"\nCombinaciones totales posibles: {total_combinations:,}")
print(f"Iteraciones a probar: 50 (RandomizedSearchCV)")
```

```
[1/5] Definiendo espacio de búsqueda de hiperparámetros...

Hiperparámetros a optimizar:
  - n_estimators: [50, 100, 150, 200, 300]
  - max_depth: [5, 8, 10, 12, 15, 20, None]
  - min_samples_split: [2, 5, 10, 15, 20, 30, 50]
  - min_samples_leaf: [1, 2, 5, 10, 15, 20, 30]
  - max_features: ['sqrt', 'log2', 0.3, 0.5, 0.7]
  - bootstrap: [True, False]
  - criterion: ['gini', 'entropy']

Combinaciones totales posibles: 34,300
Iteraciones a probar: 50 (RandomizedSearchCV)
```

In [36]:

```python
# ==========================================================================
# BÚSQUEDA DE HIPERPARÁMETROS CON RANDOMIZED SEARCH CV
# ==========================================================================
print("\n[2/5] Ejecutando RandomizedSearchCV...")
print("NOTA: Este proceso puede tardar varios minutos...")

# Modelo base
rf_base = RandomForestClassifier(
    class_weight='balanced',
    random_state=RANDOM_STATE,
    n_jobs=-1
)

# RandomizedSearchCV con validación cruzada
# IMPORTANTE: n_jobs=1 para evitar errores de multiprocessing en Jupyter not
random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions,
    n_iter=30,  # Reducido a 30 para acelerar
    cv=3,       # 3-fold CV para acelerar
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=1,   # Evita FileNotFoundError en notebooks
    verbose=2,
    return_train_score=True
)

# Ejecutar búsqueda
import time
start_time = time.time()

random_search.fit(X2_train, y2_train)

elapsed_time = time.time() - start_time
print(f"\n¡Búsqueda completada en {elapsed_time/60:.2f} minutos!")
```

```
[2/5] Ejecutando RandomizedSearchCV...
NOTA: Este proceso puede tardar varios minutos...
Fitting 3 folds for each of 30 candidates, totalling 90 fits
[CV] END bootstrap=True, criterion=entropy, max_depth=20, max_features=0.7,
min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=  13.
1s
[CV] END bootstrap=True, criterion=entropy, max_depth=20, max_features=0.7,
min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=  12.
2s
[CV] END bootstrap=True, criterion=entropy, max_depth=20, max_features=0.7,
min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=  12.
0s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=10, min_samples_split=20, n_estimators=50; total time=   2.5s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=10, min_samples_split=20, n_estimators=50; total time=   2.8s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=10, min_samples_split=20, n_estimators=50; total time=   2.7s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  14.
1s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  12.
7s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  12.
8s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=sqrt, mi
n_samples_leaf=15, min_samples_split=2, n_estimators=50; total time=   2.7s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=sqrt, mi
n_samples_leaf=15, min_samples_split=2, n_estimators=50; total time=   2.7s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=sqrt, mi
n_samples_leaf=15, min_samples_split=2, n_estimators=50; total time=   2.9s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=0.
5, min_samples_leaf=20, min_samples_split=15, n_estimators=50; total time=
8.5s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=0.
5, min_samples_leaf=20, min_samples_split=15, n_estimators=50; total time=
8.4s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=0.
5, min_samples_leaf=20, min_samples_split=15, n_estimators=50; total time=
8.8s
[CV] END bootstrap=False, criterion=gini, max_depth=12, max_features=0.7, mi
n_samples_leaf=15, min_samples_split=15, n_estimators=150; total time=  35.5
s
[CV] END bootstrap=False, criterion=gini, max_depth=12, max_features=0.7, mi
n_samples_leaf=15, min_samples_split=15, n_estimators=150; total time=  34.0
s
[CV] END bootstrap=False, criterion=gini, max_depth=12, max_features=0.7, mi
n_samples_leaf=15, min_samples_split=15, n_estimators=150; total time=  34.1
s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=sqr
t, min_samples_leaf=5, min_samples_split=30, n_estimators=200; total time=
14.6s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=sqr
t, min_samples_leaf=5, min_samples_split=30, n_estimators=200; total time=
```

```
14.7s
[CV] END bootstrap=True, criterion=entropy, max_depth=None, max_features=sqr
t, min_samples_leaf=5, min_samples_split=30, n_estimators=200; total time=
15.8s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=log2, min
_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   2.1s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=log2, min
_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   1.8s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=log2, min
_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   1.8s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=1, min_samples_split=50, n_estimators=300; total time=  17.3s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=1, min_samples_split=50, n_estimators=300; total time=  16.4s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=0.5, min_
samples_leaf=1, min_samples_split=50, n_estimators=300; total time=  16.8s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.7, min_
samples_leaf=30, min_samples_split=15, n_estimators=200; total time=  22.3s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.7, min_
samples_leaf=30, min_samples_split=15, n_estimators=200; total time=  22.2s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.7, min_
samples_leaf=30, min_samples_split=15, n_estimators=200; total time=  21.8s
[CV] END bootstrap=True, criterion=gini, max_depth=15, max_features=log2, mi
n_samples_leaf=15, min_samples_split=30, n_estimators=100; total time=   5.0
s
[CV] END bootstrap=True, criterion=gini, max_depth=15, max_features=log2, mi
n_samples_leaf=15, min_samples_split=30, n_estimators=100; total time=   5.1
s
[CV] END bootstrap=True, criterion=gini, max_depth=15, max_features=log2, mi
n_samples_leaf=15, min_samples_split=30, n_estimators=100; total time=   5.2
s
[CV] END bootstrap=False, criterion=gini, max_depth=5, max_features=log2, mi
n_samples_leaf=15, min_samples_split=50, n_estimators=200; total time=   5.8
s
[CV] END bootstrap=False, criterion=gini, max_depth=5, max_features=log2, mi
n_samples_leaf=15, min_samples_split=50, n_estimators=200; total time=   5.9
s
[CV] END bootstrap=False, criterion=gini, max_depth=5, max_features=log2, mi
n_samples_leaf=15, min_samples_split=50, n_estimators=200; total time=   5.9
s
[CV] END bootstrap=False, criterion=gini, max_depth=10, max_features=sqrt, m
in_samples_leaf=15, min_samples_split=30, n_estimators=300; total time=  17.
2s
[CV] END bootstrap=False, criterion=gini, max_depth=10, max_features=sqrt, m
in_samples_leaf=15, min_samples_split=30, n_estimators=300; total time=  17.
2s
[CV] END bootstrap=False, criterion=gini, max_depth=10, max_features=sqrt, m
in_samples_leaf=15, min_samples_split=30, n_estimators=300; total time=  17.
3s
[CV] END bootstrap=False, criterion=entropy, max_depth=10, max_features=0.3,
min_samples_leaf=1, min_samples_split=30, n_estimators=200; total time=  22.
2s
[CV] END bootstrap=False, criterion=entropy, max_depth=10, max_features=0.3,
min_samples_leaf=1, min_samples_split=30, n_estimators=200; total time=  22.
8s
[CV] END bootstrap=False, criterion=entropy, max_depth=10, max_features=0.3,
```

```
min_samples_leaf=1, min_samples_split=30, n_estimators=200; total time=  22.
4s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=log2, mi
n_samples_leaf=1, min_samples_split=30, n_estimators=100; total time=   6.4s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=log2, mi
n_samples_leaf=1, min_samples_split=30, n_estimators=100; total time=   6.7s
[CV] END bootstrap=True, criterion=gini, max_depth=20, max_features=log2, mi
n_samples_leaf=1, min_samples_split=30, n_estimators=100; total time=   6.5s
[CV] END bootstrap=False, criterion=entropy, max_depth=8, max_features=0.3,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   5.2s
[CV] END bootstrap=False, criterion=entropy, max_depth=8, max_features=0.3,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   5.3s
[CV] END bootstrap=False, criterion=entropy, max_depth=8, max_features=0.3,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   5.0s
[CV] END bootstrap=False, criterion=gini, max_depth=None, max_features=0.7,
min_samples_leaf=20, min_samples_split=2, n_estimators=200; total time= 1.1m
in
[CV] END bootstrap=False, criterion=gini, max_depth=None, max_features=0.7,
min_samples_leaf=20, min_samples_split=2, n_estimators=200; total time= 1.1m
in
[CV] END bootstrap=False, criterion=gini, max_depth=None, max_features=0.7,
min_samples_leaf=20, min_samples_split=2, n_estimators=200; total time= 1.1m
in
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.3, min
_samples_leaf=5, min_samples_split=5, n_estimators=150; total time=  13.5s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.3, min
_samples_leaf=5, min_samples_split=5, n_estimators=150; total time=  12.3s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.3, min
_samples_leaf=5, min_samples_split=5, n_estimators=150; total time=  12.4s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=sqrt, mi
n_samples_leaf=2, min_samples_split=20, n_estimators=100; total time=   4.7s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=sqrt, mi
n_samples_leaf=2, min_samples_split=20, n_estimators=100; total time=   4.7s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=sqrt, mi
n_samples_leaf=2, min_samples_split=20, n_estimators=100; total time=   4.8s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=log2, mi
n_samples_leaf=2, min_samples_split=15, n_estimators=150; total time=   5.9s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=log2, mi
n_samples_leaf=2, min_samples_split=15, n_estimators=150; total time=   6.1s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=log2, mi
n_samples_leaf=2, min_samples_split=15, n_estimators=150; total time=   5.9s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=sqrt, min
_samples_leaf=20, min_samples_split=10, n_estimators=300; total time=   6.6s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=sqrt, min
_samples_leaf=20, min_samples_split=10, n_estimators=300; total time=   6.4s
[CV] END bootstrap=True, criterion=gini, max_depth=5, max_features=sqrt, min
_samples_leaf=20, min_samples_split=10, n_estimators=300; total time=   6.5s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.5, min
_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=  25.4s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.5, min
_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=  25.9s
[CV] END bootstrap=False, criterion=gini, max_depth=8, max_features=0.5, min
_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=  25.7s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.3, min_
samples_leaf=20, min_samples_split=5, n_estimators=300; total time=  16.6s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.3, min_
```

```
samples_leaf=20, min_samples_split=5, n_estimators=300; total time=  16.3s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=0.3, min_
samples_leaf=20, min_samples_split=5, n_estimators=300; total time=  16.6s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=sqrt, min
_samples_leaf=2, min_samples_split=5, n_estimators=150; total time=   5.2s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=sqrt, min
_samples_leaf=2, min_samples_split=5, n_estimators=150; total time=   5.1s
[CV] END bootstrap=True, criterion=gini, max_depth=8, max_features=sqrt, min
_samples_leaf=2, min_samples_split=5, n_estimators=150; total time=   5.1s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=0.5,
min_samples_leaf=20, min_samples_split=15, n_estimators=100; total time=  2
6.4s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=0.5,
min_samples_leaf=20, min_samples_split=15, n_estimators=100; total time=  2
7.2s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=0.5,
min_samples_leaf=20, min_samples_split=15, n_estimators=100; total time=  2
8.4s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=10, min_samples_split=15, n_estimators=300; total time=  1
5.1s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=10, min_samples_split=15, n_estimators=300; total time=  1
4.8s
[CV] END bootstrap=True, criterion=entropy, max_depth=10, max_features=log2,
min_samples_leaf=10, min_samples_split=15, n_estimators=300; total time=  1
4.6s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=0.7, min
_samples_leaf=10, min_samples_split=20, n_estimators=100; total time=  15.2s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=0.7, min
_samples_leaf=10, min_samples_split=20, n_estimators=100; total time=  15.4s
[CV] END bootstrap=True, criterion=gini, max_depth=10, max_features=0.7, min
_samples_leaf=10, min_samples_split=20, n_estimators=100; total time=  16.1s
[CV] END bootstrap=True, criterion=gini, max_depth=12, max_features=sqrt, mi
n_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   2.6s
[CV] END bootstrap=True, criterion=gini, max_depth=12, max_features=sqrt, mi
n_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   2.5s
[CV] END bootstrap=True, criterion=gini, max_depth=12, max_features=sqrt, mi
n_samples_leaf=30, min_samples_split=5, n_estimators=50; total time=   2.7s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=sqr
t, min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=
5.1s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=sqr
t, min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=
5.1s
[CV] END bootstrap=False, criterion=entropy, max_depth=15, max_features=sqr
t, min_samples_leaf=10, min_samples_split=10, n_estimators=50; total time=
5.1s
[CV] END bootstrap=True, criterion=entropy, max_depth=15, max_features=0.7,
min_samples_leaf=2, min_samples_split=10, n_estimators=150; total time=  34.
6s
[CV] END bootstrap=True, criterion=entropy, max_depth=15, max_features=0.7,
min_samples_leaf=2, min_samples_split=10, n_estimators=150; total time=  37.
5s
[CV] END bootstrap=True, criterion=entropy, max_depth=15, max_features=0.7,
min_samples_leaf=2, min_samples_split=10, n_estimators=150; total time=  38.
```

5s

¡Búsqueda completada en 22.75 minutos!

In [48]:
```
# ===========================================================================
# MEJORES HIPERPARÁMETROS ENCONTRADOS
# ===========================================================================
print("\n[3/5] Mejores hiperparámetros encontrados:")
print("="*80)

best_params = random_search.best_params_
best_score = random_search.best_score_

print(f"\nMejor ROC-AUC en CV: {best_score:.4f}")
print(f"\nHiperparámetros óptimos:")
for param, value in best_params.items():
    print(f"  {param}: {value}")

# Mostrar top 10 combinaciones
print("\n" + "="*80)
print("Top 10 mejores combinaciones de hiperparámetros:")
print("="*80)

cv_results = pd.DataFrame(random_search.cv_results_)
cv_results_sorted = cv_results.sort_values('rank_test_score').head(10)

for idx, row in cv_results_sorted.iterrows():
    print(f"\nRank {row['rank_test_score']:.0f}: ROC-AUC = {row['mean_test_s
    print(f"  Train AUC: {row['mean_train_score']:.4f}, Overfitting: {row['m
    params = row['params']
    for p, v in params.items():
        print(f"    {p}: {v}")
```

```
[3/5] Mejores hiperparámetros encontrados:
============================================================================
====

Mejor ROC-AUC en CV: 0.7448

Hiperparámetros óptimos:
  n_estimators: 200
  min_samples_split: 30
  min_samples_leaf: 5
  max_features: sqrt
  max_depth: None
  criterion: entropy
  bootstrap: True

============================================================================
====
Top 10 mejores combinaciones de hiperparámetros:
============================================================================
====

Rank 1: ROC-AUC = 0.7448 (+/- 0.0020)
  Train AUC: 0.9909, Overfitting: 0.2461
    n_estimators: 200
    min_samples_split: 30
    min_samples_leaf: 5
    max_features: sqrt
    max_depth: None
    criterion: entropy
    bootstrap: True

Rank 2: ROC-AUC = 0.7432 (+/- 0.0014)
  Train AUC: 0.8241, Overfitting: 0.0809
    n_estimators: 50
    min_samples_split: 5
    min_samples_leaf: 30
    max_features: sqrt
    max_depth: 12
    criterion: gini
    bootstrap: True

Rank 3: ROC-AUC = 0.7429 (+/- 0.0015)
  Train AUC: 0.8944, Overfitting: 0.1515
    n_estimators: 100
    min_samples_split: 30
    min_samples_leaf: 15
    max_features: log2
    max_depth: 15
    criterion: gini
    bootstrap: True

Rank 4: ROC-AUC = 0.7429 (+/- 0.0018)
  Train AUC: 0.8136, Overfitting: 0.0707
    n_estimators: 300
    min_samples_split: 30
    min_samples_leaf: 15
```

```
    max_features: sqrt
    max_depth: 10
    criterion: gini
    bootstrap: False

Rank 5: ROC-AUC = 0.7429 (+/- 0.0018)
  Train AUC: 0.8015, Overfitting: 0.0586
    n_estimators: 300
    min_samples_split: 15
    min_samples_leaf: 10
    max_features: log2
    max_depth: 10
    criterion: entropy
    bootstrap: True

Rank 6: ROC-AUC = 0.7421 (+/- 0.0018)
  Train AUC: 0.8049, Overfitting: 0.0628
    n_estimators: 300
    min_samples_split: 10
    min_samples_leaf: 1
    max_features: log2
    max_depth: 10
    criterion: entropy
    bootstrap: True

Rank 7: ROC-AUC = 0.7418 (+/- 0.0020)
  Train AUC: 0.7731, Overfitting: 0.0313
    n_estimators: 300
    min_samples_split: 5
    min_samples_leaf: 20
    max_features: 0.3
    max_depth: 8
    criterion: gini
    bootstrap: True

Rank 8: ROC-AUC = 0.7415 (+/- 0.0018)
  Train AUC: 0.8078, Overfitting: 0.0664
    n_estimators: 150
    min_samples_split: 15
    min_samples_leaf: 2
    max_features: log2
    max_depth: 10
    criterion: gini
    bootstrap: True

Rank 9: ROC-AUC = 0.7415 (+/- 0.0016)
  Train AUC: 0.9424, Overfitting: 0.2009
    n_estimators: 50
    min_samples_split: 2
    min_samples_leaf: 15
    max_features: sqrt
    max_depth: 20
    criterion: gini
    bootstrap: True

Rank 10: ROC-AUC = 0.7414 (+/- 0.0025)
```

```
Train AUC: 0.8172, Overfitting: 0.0758
  n_estimators: 200
  min_samples_split: 30
  min_samples_leaf: 1
  max_features: 0.3
  max_depth: 10
  criterion: entropy
  bootstrap: False
```

In [41]:

```
# ============================================================================
# EVALUACIÓN DEL MODELO OPTIMIZADO
# ============================================================================
print("\n[4/5] Evaluando modelo con hiperparámetros óptimos...")
print("="*80)

# El mejor modelo ya está entrenado
best_model = random_search.best_estimator_

# Predicciones
y3_train_pred = best_model.predict(X2_train)
y3_train_proba = best_model.predict_proba(X2_train)[:, 1]

y3_test_pred = best_model.predict(X2_test)
y3_test_proba = best_model.predict_proba(X2_test)[:, 1]

# Métricas en Train
train3_auc = roc_auc_score(y2_train, y3_train_proba)
train3_ap = average_precision_score(y2_train, y3_train_proba)

# Métricas en Test
test3_auc = roc_auc_score(y2_test, y3_test_proba)
test3_ap = average_precision_score(y2_test, y3_test_proba)

print(f"\n{'='*80}")
print("RESULTADOS DEL MODELO OPTIMIZADO")
print(f"{'='*80}")

print(f"\n{'Metrica':<25} {'Train':<12} {'Test':<12} {'Diferencia':<12}")
print("-" * 65)
print(f"{'ROC-AUC':<25} {train3_auc:<12.4f} {test3_auc:<12.4f} {abs(train3_a
print(f"{'Average Precision':<25} {train3_ap:<12.4f} {test3_ap:<12.4f} {abs(

print(f"\nClassification Report (Test Set):")
print(classification_report(y2_test, y3_test_pred,
                            target_names=['Pago (0)', 'Default (1)']))

print(f"\nConfusion Matrix (Test Set):")
cm3_test = confusion_matrix(y2_test, y3_test_pred)
print(cm3_test)
print(f"\nInterpretacion:")
print(f"  Verdaderos Negativos (TN): {cm3_test[0,0]:,}")
print(f"  Falsos Positivos (FP): {cm3_test[0,1]:,}")
print(f"  Falsos Negativos (FN): {cm3_test[1,0]:,}")
print(f"  Verdaderos Positivos (TP): {cm3_test[1,1]:,}")
```

```
[4/5] Evaluando modelo con hiperparámetros óptimos...
============================================================================
====

============================================================================
====
RESULTADOS DEL MODELO OPTIMIZADO
============================================================================
====

Metrica                   Train        Test         Diferencia
-----------------------------------------------------------------
ROC-AUC                   0.9905       0.7482       0.2423
Average Precision         0.8949       0.2281       0.6668

Classification Report (Test Set):
              precision    recall  f1-score   support

    Pago (0)       0.93      0.95      0.94     56538
 Default (1)       0.29      0.24      0.26      4965

    accuracy                           0.89     61503
   macro avg       0.61      0.59      0.60     61503
weighted avg       0.88      0.89      0.89     61503


Confusion Matrix (Test Set):
[[53668  2870]
 [ 3789  1176]]

Interpretacion:
  Verdaderos Negativos (TN): 53,668
  Falsos Positivos (FP): 2,870
  Falsos Negativos (FN): 3,789
  Verdaderos Positivos (TP): 1,176
```

In [44]:

```
# ============================================================================
# COMPARACIÓN: MODELO 2 ORIGINAL VS MODELO OPTIMIZADO
# ============================================================================

# Extraer mejores hiperparámetros (si no están definidos)
best_params = random_search.best_params_

print("\n" + "="*80)
print("COMPARACIÓN: RF2 ORIGINAL vs RF3 OPTIMIZADO")
print("="*80)

print(f"\n{'Metrica':<25} {'RF2 Original':<15} {'RF3 Optimizado':<15} {'Mejo
print("-" * 70)
print(f"{'ROC-AUC (Test)':<25} {test2_auc:<15.4f} {test3_auc:<15.4f} {test3_
print(f"{'Average Precision (Test)':<25} {test2_ap:<15.4f} {test3_ap:<15.4f}
print(f"{'ROC-AUC (Train)':<25} {train2_auc:<15.4f} {train3_auc:<15.4f} {tra
print(f"{'Overfitting (Gap)':<25} {train2_auc - test2_auc:<15.4f} {train3_au

print("\n" + "="*80)
print("HIPERPARÁMETROS: ORIGINAL vs OPTIMIZADO")
```

```python
print("="*80)
print(f"\n{'Parámetro':<25} {'RF2 Original':<20} {'RF3 Optimizado':<20}")
print("-" * 65)
print(f"{'n_estimators':<25} {N_ESTIMATORS:<20} {best_params.get('n_estimato
print(f"{'max_depth':<25} {MAX_DEPTH:<20} {str(best_params.get('max_depth',
print(f"{'min_samples_split':<25} {MIN_SAMPLES_SPLIT:<20} {best_params.get('
print(f"{'min_samples_leaf':<25} {MIN_SAMPLES_LEAF:<20} {best_params.get('mi
print(f"{'max_features':<25} {'auto':<20} {str(best_params.get('max_features
print(f"{'criterion':<25} {'gini':<20} {best_params.get('criterion', 'gini')
print(f"{'bootstrap':<25} {'True':<20} {str(best_params.get('bootstrap', Tru

# Análisis de mejora
diff_auc = test3_auc - test2_auc
diff_overfitting = (train3_auc - test3_auc) - (train2_auc - test2_auc)

print("\n" + "="*80)
print("ANÁLISIS DE MEJORA")
print("="*80)
if diff_auc > 0:
    print(f"\n✓ El modelo optimizado MEJORA el ROC-AUC en {diff_auc:.4f} ({d
else:
    print(f"\n✗ El modelo optimizado NO mejora el ROC-AUC ({diff_auc:.4f})")

if diff_overfitting < 0:
    print(f"✓ El overfitting se REDUCE en {abs(diff_overfitting):.4f}")
else:
    print(f"✗ El overfitting AUMENTA en {diff_overfitting:.4f}")
```

```
================================================================================
====
COMPARACIÓN: RF2 ORIGINAL vs RF3 OPTIMIZADO
================================================================================
====

Metrica                   RF2 Original    RF3 Optimizado  Mejora
-------------------------------------------------------------------------
ROC-AUC (Test)            0.7407          0.7482          +0.0074
Average Precision (Test)  0.2133          0.2281          +0.0148
ROC-AUC (Train)           0.9038          0.9905          +0.0867
Overfitting (Gap)         0.1631          0.2423          +0.0792

================================================================================
====
HIPERPARÁMETROS: ORIGINAL vs OPTIMIZADO
================================================================================
====

Parámetro                 RF2 Original         RF3 Optimizado
------------------------------------------------------------------
n_estimators              100                  200
max_depth                 15                   None
min_samples_split         10                   30
min_samples_leaf          5                    5
max_features              auto                 sqrt
criterion                 gini                 entropy
bootstrap                 True                 True

================================================================================
====
ANÁLISIS DE MEJORA
================================================================================
====

✓ El modelo optimizado MEJORA el ROC-AUC en 0.0074 (0.74%)
✗ El overfitting AUMENTA en 0.0792
```

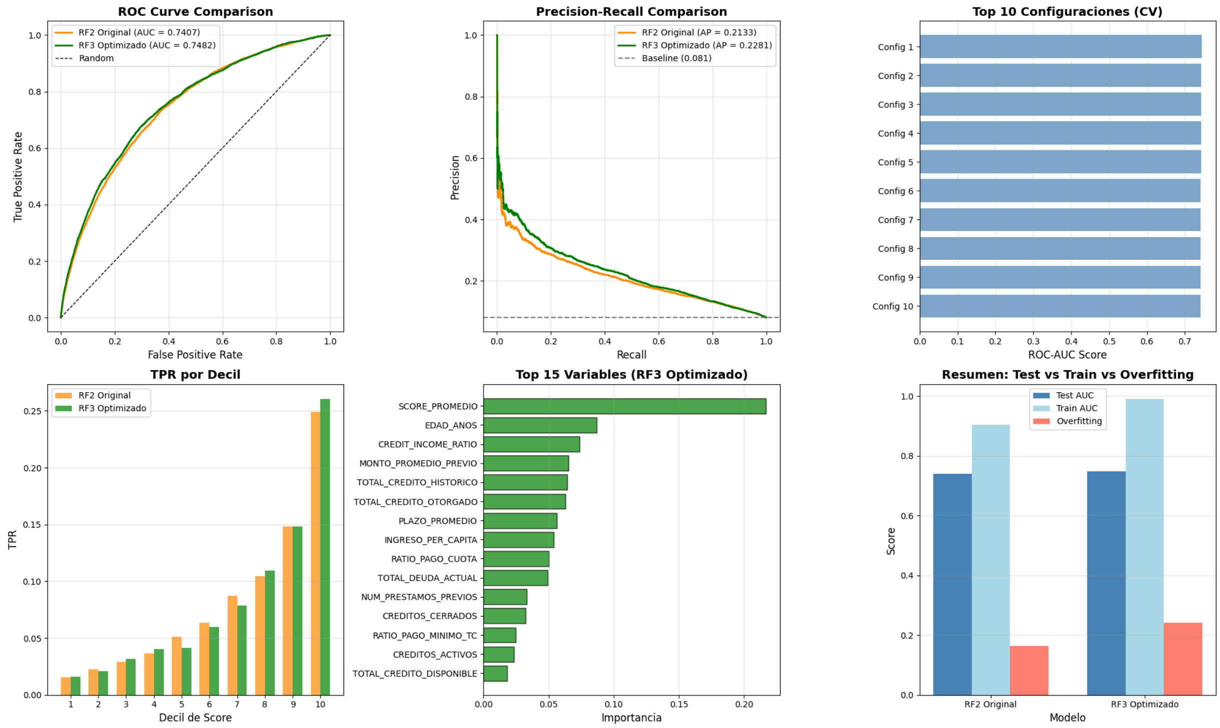
In [46]:

```python
# ============================================================================
# VISUALIZACIONES DEL MODELO OPTIMIZADO
# ============================================================================
print("\n[5/5] Generando visualizaciones del modelo optimizado...")

# Extraer resultados de CV (si no están definidos)
cv_results = pd.DataFrame(random_search.cv_results_)
cv_results_sorted = cv_results.sort_values('rank_test_score').head(10)

# Calcular deciles para modelo optimizado
test_results3 = pd.DataFrame({
    'y_true': y2_test,
    'y_proba': y3_test_proba
})
test_results3['decil'] = pd.qcut(test_results3['y_proba'], q=10, labels=Fals

decil_stats3 = test_results3.groupby('decil').agg({
    'y_true': ['sum', 'count', 'mean']
```

```python
}).reset_index()
decil_stats3.columns = ['decil', 'positivos', 'total', 'tpr']
decil_stats3 = decil_stats3.sort_values('decil', ascending=False)

# Feature importance del modelo optimizado
feature_importance3 = pd.DataFrame({
    'Variable': X2.columns,
    'Importancia': best_model.feature_importances_
}).sort_values('Importancia', ascending=False)

# Crear figura comparativa
fig, axes = plt.subplots(2, 3, figsize=(20, 12))

# ROC Curves comparison
fpr2, tpr2, _ = roc_curve(y2_test, y2_test_proba)
fpr3, tpr3, _ = roc_curve(y2_test, y3_test_proba)

axes[0, 0].plot(fpr2, tpr2, linewidth=2, color='darkorange', label=f'RF2 Ori
axes[0, 0].plot(fpr3, tpr3, linewidth=2, color='green', label=f'RF3 Optimiza
axes[0, 0].plot([0, 1], [0, 1], 'k--', linewidth=1, label='Random')
axes[0, 0].set_xlabel('False Positive Rate', fontsize=12)
axes[0, 0].set_ylabel('True Positive Rate', fontsize=12)
axes[0, 0].set_title('ROC Curve Comparison', fontsize=14, fontweight='bold')
axes[0, 0].legend()
axes[0, 0].grid(alpha=0.3)

# Precision-Recall comparison
precision2, recall2, _ = precision_recall_curve(y2_test, y2_test_proba)
precision3, recall3, _ = precision_recall_curve(y2_test, y3_test_proba)

axes[0, 1].plot(recall2, precision2, linewidth=2, color='darkorange', label=
axes[0, 1].plot(recall3, precision3, linewidth=2, color='green', label=f'RF3
axes[0, 1].axhline(y=y2_test.mean(), color='gray', linestyle='--', label=f'B
axes[0, 1].set_xlabel('Recall', fontsize=12)
axes[0, 1].set_ylabel('Precision', fontsize=12)
axes[0, 1].set_title('Precision-Recall Comparison', fontsize=14, fontweight=
axes[0, 1].legend()
axes[0, 1].grid(alpha=0.3)

# CV Results distribution
cv_scores_plot = cv_results_sorted['mean_test_score'].values[:10]
axes[0, 2].barh(range(10), cv_scores_plot, color='steelblue', alpha=0.7)
axes[0, 2].set_yticks(range(10))
axes[0, 2].set_yticklabels([f'Config {i+1}' for i in range(10)])
axes[0, 2].set_xlabel('ROC-AUC Score', fontsize=12)
axes[0, 2].set_title('Top 10 Configuraciones (CV)', fontsize=14, fontweight=
axes[0, 2].invert_yaxis()
axes[0, 2].grid(alpha=0.3, axis='x')

# TPR por Decil comparison
width = 0.35
x = np.arange(len(decil_stats3))
axes[1, 0].bar(x - width/2, decil_stats2.sort_values('decil')['tpr'], width,
axes[1, 0].bar(x + width/2, decil_stats3.sort_values('decil')['tpr'], width,
axes[1, 0].set_xlabel('Decil de Score', fontsize=12)
axes[1, 0].set_ylabel('TPR', fontsize=12)
```

```
axes[1, 0].set_title('TPR por Decil', fontsize=14, fontweight='bold')
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(decil_stats3.sort_values('decil')['decil'].values
axes[1, 0].legend()
axes[1, 0].grid(alpha=0.3, axis='y')

# Feature Importance (Top 15)
top_features3 = feature_importance3.head(15)
axes[1, 1].barh(range(len(top_features3)), top_features3['Importancia'].valu
axes[1, 1].set_yticks(range(len(top_features3)))
axes[1, 1].set_yticklabels(top_features3['Variable'].values)
axes[1, 1].set_xlabel('Importancia', fontsize=12)
axes[1, 1].set_title('Top 15 Variables (RF3 Optimizado)', fontsize=14, fontw
axes[1, 1].invert_yaxis()
axes[1, 1].grid(alpha=0.3, axis='x')

# Comparison summary
summary_data = {
    'Modelo': ['RF2 Original', 'RF3 Optimizado'],
    'ROC-AUC Test': [test2_auc, test3_auc],
    'ROC-AUC Train': [train2_auc, train3_auc],
    'Overfitting': [train2_auc - test2_auc, train3_auc - test3_auc]
}
summary_df = pd.DataFrame(summary_data)

colors = ['darkorange', 'green']
x_pos = np.arange(len(summary_df))
width = 0.25

bars1 = axes[1, 2].bar(x_pos - width, summary_df['ROC-AUC Test'], width, lab
bars2 = axes[1, 2].bar(x_pos, summary_df['ROC-AUC Train'], width, label='Tra
bars3 = axes[1, 2].bar(x_pos + width, summary_df['Overfitting'], width, labe

axes[1, 2].set_xlabel('Modelo', fontsize=12)
axes[1, 2].set_ylabel('Score', fontsize=12)
axes[1, 2].set_title('Resumen: Test vs Train vs Overfitting', fontsize=14, f
axes[1, 2].set_xticks(x_pos)
axes[1, 2].set_xticklabels(summary_df['Modelo'])
axes[1, 2].legend()
axes[1, 2].grid(alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(f'{OUTPUT_DIR}/random_forest3_optimized_curves.png', dpi=300, bb
print(f"Gráficas guardadas: {OUTPUT_DIR}/random_forest3_optimized_curves.png
plt.show()
```

```
[5/5] Generando visualizaciones del modelo optimizado...
Gráficas guardadas: ../../data/random-forest-output/random_forest3_optimized
_curves.png
```



In [ ]:

```
# ==========================================================================
# GUARDAR MODELO OPTIMIZADO Y RESULTADOS
# ==========================================================================
print("\nGuardando modelo optimizado y resultados...")

# Guardar modelo optimizado
with open(f'{OUTPUT_DIR}/random_forest3_optimized_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

# Guardar mejores hiperparámetros
best_params_df = pd.DataFrame([best_params])
best_params_df.to_csv(f'{OUTPUT_DIR}/best_hyperparameters.csv', index=False)

# Guardar feature importance del modelo optimizado
feature_importance3.to_csv(f'{OUTPUT_DIR}/feature_importance_rf3_optimized.c

# Guardar predicciones del modelo optimizado
test_predictions3 = pd.DataFrame({
    'SK_ID_CURR': df.iloc[X2_test.index]['SK_ID_CURR'].values,
    'TARGET_Real': y2_test.values,
    'TARGET_Predicho': y3_test_pred,
    'Probabilidad_Default': y3_test_proba
})
test_predictions3.to_csv(f'{OUTPUT_DIR}/test_predictions_rf3_optimized.csv',

# Guardar resultados de CV
cv_results.to_csv(f'{OUTPUT_DIR}/cv_search_results.csv', index=False)

print(f"\nArchivos guardados en: {OUTPUT_DIR}/")
print("  - random_forest3_optimized_model.pkl")
print("  - best_hyperparameters.csv")
print("  - feature_importance_rf3_optimized.csv")
print("  - test_predictions_rf3_optimized.csv")
```

```python
print("  - cv_search_results.csv")
print("  - random_forest3_optimized_curves.png")

# ================================================================================
# RESUMEN FINAL
# ================================================================================
print("\n" + "="*80)
print("RESUMEN FINAL - BÚSQUEDA DE HIPERPARÁMETROS COMPLETADA")
print("="*80)

print(f"\n📊 RESULTADOS DEL MODELO OPTIMIZADO (RF3)")
print(f"   ROC-AUC Test: {test3_auc:.4f}")
print(f"   Average Precision: {test3_ap:.4f}")
print(f"   Overfitting: {train3_auc - test3_auc:.4f}")

print(f"\n🎯 MEJORES HIPERPARÁMETROS:")
for param, value in best_params.items():
    print(f"   {param}: {value}")

print(f"\n📈 COMPARACIÓN CON MODELO ORIGINAL:")
print(f"   Cambio en ROC-AUC: {test3_auc - test2_auc:+.4f}")
print(f"   Cambio en Overfitting: {(train3_auc - test3_auc) - (train2_auc -

print("\n" + "="*80)
print("¡OPTIMIZACIÓN DE HIPERPARÁMETROS COMPLETADA!")
print("="*80)
```