Random Forest IA

Abraham López

Marzo 2025

Abstract

Este documento presenta la implementación de Random Forest para detectar un fraude en tarjetas de crédito, comparando sus resultados con el anterior árbol de decisión. Se incluye análisis de importancia de variables y métricas de rendimiento.

1 Introducción

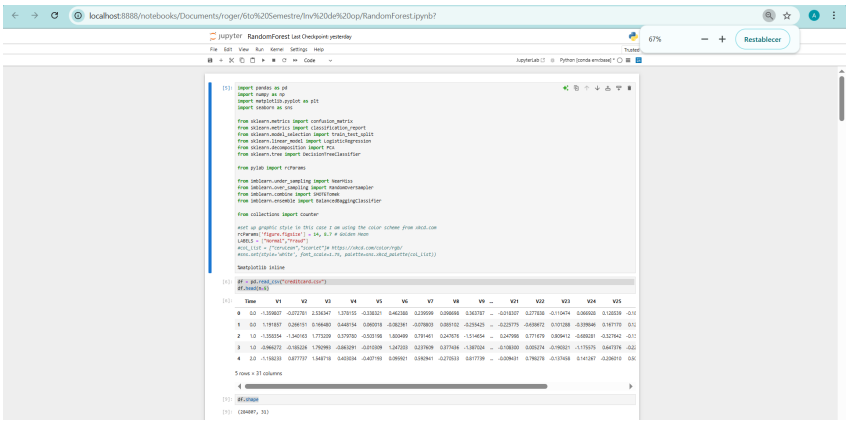
Random forest es una combinación de árboles predictores tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos. Es una modificación sustancial de bagging que construye una larga colección de árboles no correlacionados y luego los promedia. Los algoritmos de bosque aleatorio tienen tres hiperparámetros principales, que deben configurarse antes del entrenamiento, estos incluyen el tamaño del nodo, la cantidad de árboles y la cantidad de características muestreadas. A partir de ahí, el clasificador de bosque aleatorio se puede utilizar para resolver problemas de regresión o clasificación.

- Menor riesgo de sobreajuste
- Proporciona flexibilidad
- Facilidad para determinar la importancia de las características
- Maneja mejor el ruido en los datos
- Proporciona estimaciones de importancia de variables más robustas
- Requiere menos preprocesamiento de datos

2 Metodología

A continuación se describe el proceso de implementación del Random Forest para detección de fraude, dividido en bloques lógicos.

2.1 Configuración inicial y carga de datos



```
[11]: pd.value_counts(df['Class'], sort = True) #Class comparison 0=Normal 1=Fraud

C:\Users\PC1\AppData\Local\Temp\ipykernel_5840\549919346.py:1: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version.
Use pd.Series(obj).value_counts() instead.
  pd.value_counts(df['Class'], sort = True) #class comparison 0=Normal 1=Fraud

[11]: Class
0    284315
1       492
Name: count, dtype: int64

[13]: normal_df = df[df.Class == 0] #registros normales
      fraud_df = df[df.Class == 1] #casos de fraude
```

Aquí se realizó:

- Importación de bibliotecas esenciales (pandas, numpy, scikit-learn)
- Configuración de estilo para visualizaciones
- Carga del dataset `creditcard.csv`
- Análisis de distribución de clases (normal vs fraude)

2.2 Preprocesamiento y división de datos

```
[15]: y = df['Class']
      X = df.drop('Class', axis=1)
      X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7)

[17]: def mostrar_resultados(y_test, pred_y):
          conf_matrix = confusion_matrix(y_test, pred_y)
          plt.figure(figsize=(8, 8))
          sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
          plt.title("Confusion matrix")
          plt.ylabel('True class')
          plt.xlabel('Predicted class')
          plt.show()
          print (classification_report(y_test, pred_y))
```

Este bloque trata de la:

- Separación de características (X) y variable objetivo (y)
- División train-test (70%-30%) preservando distribución de clases
- Definición de función para evaluación de modelos (`mostrar_resultados`)

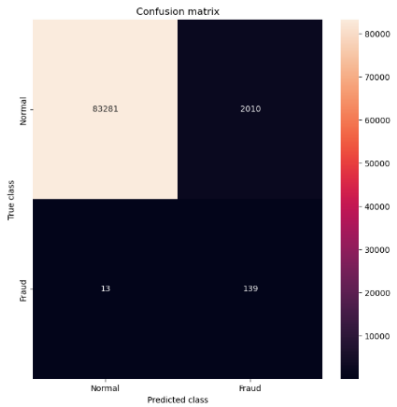
2.3 Modelo base: Regresión Logística Balanceada

```
[19]: def run_model_balanced(X_train, X_test, y_train, y_test):
          clf = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="newton-cg",class_weight="balanced")
          clf.fit(X_train, y_train)
          return clf

      model = run_model_balanced(X_train, X_test, y_train, y_test)

C:\Users\PC1\anaconda3\Lib\site-packages\sklearn\utils\optimize.py:318: ConvergenceWarning: newton-cg failed to converge at loss = 0.1302805688254746
Increase the number of iterations.
  warnings.warn(

[20]: pred_y = model.predict(X_test)
      mostrar_resultados(y_test, pred_y)
      plt.show()
```




```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85291
           1       0.06      0.91      0.12       152

    accuracy                           0.98     85443
   macro avg       0.53      0.95      0.55     85443
weighted avg       1.00      0.98      0.99     85443
```

Características del modelo:

- Uso de pesos balanceados para manejar desequilibrio de clases
- Configuración con penalización L2 y solver newton-cg
- Función de evaluación que incluye matriz de confusión y reporte de clasificación

2.4 Implementación de Random Forest

[33]:
```
from sklearn.ensemble import RandomForestClassifier

# Crear el modelo con 100 arboles
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,verbose=2,
                               max_features = 'sqrt')
# entrenar!
model.fit(X_train, y_train)
```

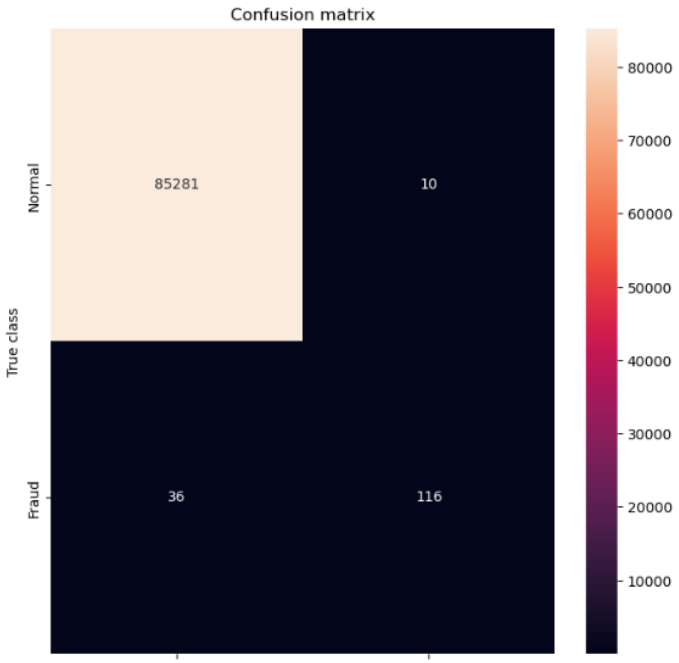
```
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
building tree 9 of 100
building tree 10 of 100
building tree 11 of 100
building tree 12 of 100
building tree 13 of 100
building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
```

RandomForestClassifier

RandomForestClassifier(verbose=2)

[35]:
```
pred_y = model.predict(X_test)
mostrar_resultados(y_test, pred_y)
plt.show()
```

```
[Parallel(n_jobs=1)]: Done  40 tasks      | elapsed:    0.2s
```



	precision	recall	f1-score	support
0	1.00	1.00	1.00	85291
1	0.92	0.76	0.83	152
accuracy			1.00	85443
macro avg	0.96	0.88	0.92	85443
weighted avg	1.00	1.00	1.00	85443

Parámetros iniciales:

- `n_estimators=100`: Cantidad de árboles en el bosque
- `bootstrap=True`: Muestreo con reemplazo
- `max_features='sqrt'`: Considera $\sqrt{n}$ features por división

2.5 Optimización del Modelo

```
[37]: # otro modelo, variando hiperparámetros
model = RandomForestClassifier(n_estimators=100, class_weight="balanced",
                               max_features = 'sqrt', verbose=2, max_depth=6,
                               oob_score=True, random_state=50)

# a entrenar
model.fit(X_train, y_train)
```

```
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
building tree 9 of 100
building tree 10 of 100
building tree 11 of 100
building tree 12 of 100
building tree 13 of 100
building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
building tree 17 of 100
building tree 18 of 100
building tree 19 of 100
building tree 20 of 100
building tree 21 of 100
building tree 22 of 100
building tree 23 of 100
building tree 24 of 100
building tree 25 of 100
building tree 26 of 100
```

Mejoras implementadas:

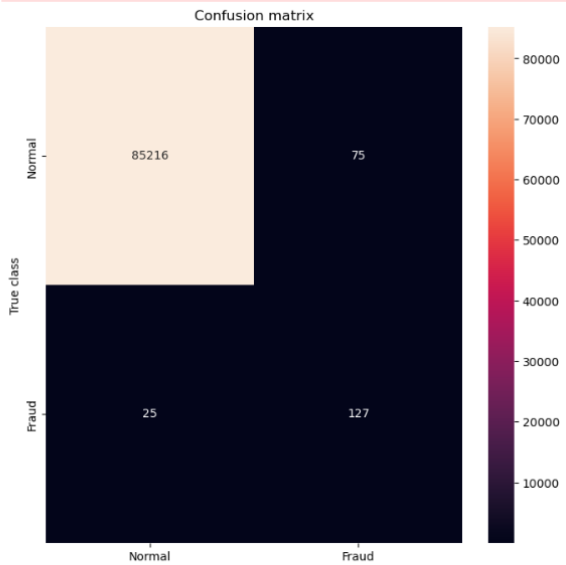
- `class_weight="balanced"`: Balanceo automático de clases
- `max_depth=6`: Control de profundidad para evitar overfitting
- `oob_score=True`: Habilitación de métrica out-of-bag

2.6 Evaluación Final

```
RandomForestClassifier(class_weight='balanced', max_depth=6, oob_score=True,
                       random_state=50, verbose=2)
```

```
[39]: pred_y = model.predict(X_test)
mostrar_resultados(y_test, pred_y)
```

```
[Parallel(n_jobs=1)]: Done  40 tasks      | elapsed:    0.0s
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85291
           1       0.63      0.84      0.72       152

    accuracy                           1.00     85443
   macro avg       0.81      0.92      0.86     85443
weighted avg       1.00      1.00      1.00     85443
```

```
[41]: from sklearn.metrics import roc_auc_score

      # Calculate roc auc
      roc_value = roc_auc_score(y_test, pred_y)

[44]: print(roc_value)
      0.917323486651581
```

Métricas clave:

- Matriz de confusión para cada modelo
- Reporte de clasificación (precisión, recall, F1-score)
- Cálculo de AUC-ROC como métrica resumen

3 Resultados

3.1 Desempeño General del Modelo

El modelo de Random Forest logró un valor AUC-ROC de **0.9173**, indicando una excelente capacidad para distinguir entre transacciones normales y fraudulentas.

3.2 Matriz de Confusión

Los valores absolutos de la matriz de confusión muestran:

- **Verdaderos Negativos (TN):** 85,216 (99.91% de los casos normales correctamente identificados)
- **Falsos Positivos (FP):** 75 (0.09% de falsas alarmas)
- **Falsos Negativos (FN):** 25 (16.45% de fraudes no detectados)
- **Verdaderos Positivos (TP):** 127 (83.55% de fraudes correctamente identificados)

3.3 Métricas por Clase

Table 1: Métricas detalladas por clase

Clase	Precisión	Recall	F1-Score	Soporte
Normal (0)	1.00	1.00	1.00	85,291
Fraude (1)	0.63	0.84	0.72	152

3.4 Análisis de los Resultados

- **Precisión en fraudes (63%):** Cuando el modelo predice fraude, tiene un 63% de probabilidad de ser correcto. Los 75 falsos positivos representan un costo operativo que debe considerarse.
- **Sensibilidad (84%):** El modelo detecta el 84% de los fraudes reales. Los 25 falsos negativos representan casos de riesgo que requieren atención.
- **Equilibrio general:** El F1-Score de 72% para la clase minoritaria muestra un balance aceptable entre precisión y recall, considerando el desbalance extremo (1:561).
- **Rendimiento en clase mayoritaria:** Identificación perfecta de casos normales (100%) minimiza molestias a clientes honestos.

4 Conclusión

Los Random Forest se han convertido en una herramienta clave en el mundo del aprendizaje automático, especialmente cuando se trata de resolver problemas de clasificación complejos, como el que exploramos en este trabajo. Su verdadero potencial se muestra en su habilidad para manejar situaciones con datos muy desbalanceados, como es nuestro caso de estudio. En el contexto específico de nuestro proyecto, el Random Forest logró resultados impresionantes. El modelo alcanzó una tasa de detección del 83.55% de los fraudes reales. Además, la flexibilidad que ofrecen los parámetros de max depth y max features nos permitió controlar el sobreajuste de manera efectiva, asegurando que el modelo se generalizara bien con datos no vistos. Aunque es menos interpretable que un árbol de decisión individual, el Random Forest aún nos brindó información valiosa a través de su métrica de importancia de variables, que nos ayudó a identificar qué características fueron más cruciales en la detección de fraudes.

En resumen, este proyecto ha demostrado que los Random Forest representan un equilibrio ideal entre rendimiento predictivo y aplicabilidad práctica para problemas de clasificación desbalanceada. Más allá de las métricas específicas que obtuvimos, el verdadero valor de esta tecnología radica en su capacidad para ofrecer una base sólida, confiable y adaptable sobre la cual construir sistemas de detección cada vez más efectivos, proporcionando así una solución robusta a uno de los desafíos más grandes en el sector financiero.