Predicción de quiebra bancaria taiwanesa mediante Clasificadores Bayesianos, Estimadores No Paramétricos y k-NN

Jordi Blasco Lozano 74527208D

3 de noviembre de 2025

Resumen

He abordado la práctica 2 seleccionando el dataset "Taiwanese Bankruptcy Prediction" de UCI, aplicando los métodos estudiados en la asignatura. Explico cada proceso (preprocesado, partición, modelado, optimización y análisis de resultados) usando terminaciones verbales en primera persona, incluyendo los códigos implementados para que cada apartado sea reproducible y comprensible.

1 Introducción

He decidido realizar la práctica usando el dataset "Taiwanese Bankruptcy Prediction"porque corresponde a un problema clásico de clasificación binaria con fuerte desbalance y datos financieros reales. Este dataset presenta características que lo hacen especialmente interesante para evaluar clasificadores:

- **Alta dimensionalidad:** 95 características financieras que describen la salud económica de empresas
- **Desbalance severo:** Solo el 3.23 % de las empresas están en bancarrota, lo que representa un reto significativo para los algoritmos de clasificación
- **Relevancia práctica:** La predicción de quiebras empresariales tiene aplicaciones directas en análisis de riesgo crediticio y decisiones de inversión
- **Datos reales:** Proviene de empresas taiwanesas reales, lo que añade complejidad y ruido natural

El objetivo principal es comparar el rendimiento de diferentes enfoques de clasificación (paramétricos vs. no paramétricos) bajo condiciones de desbalance extremo, poniendo especial énfasis en la correcta aplicación de técnicas de validación cruzada para evitar resultados optimistas y engañosos.

2 Dataset y Preprocesado

He descargado el dataset desde el repositorio UCI Machine Learning. El conjunto contiene 6819 empresas taiwanesas con 95 características financieras cada una. Tras el análisis exploratorio he identificado las siguientes propiedades:

2.1 Características del Dataset

- **Tamaño:** 6819 ejemplos con 96 columnas (95 features + 1 target)
- **Tipos de datos:** 93 variables float64 (continuas) y 3 int64 (flags binarios)
- **Valores faltantes:** Ninguno (100 % de datos completos)
- **Distribución de clases:** 220 bancarrotas (3.23 %) vs 6599 no bancarrotas (96.77 %)
- **Ratio de desbalance:** Aproximadamente 30:1 (clase mayoritaria:minoritaria)

Este desbalance extremo es un desafío fundamental que afectará significativamente al entrenamiento y evaluación de los modelos. Los clasificadores tienden naturalmente a favorecer la clase mayoritaria, lo que puede llevar a modelos con alta accuracy pero pobre capacidad de detectar bancarrotas.

2.2 Preprocesamiento

No he realizado normalización ni estandarización en esta fase inicial porque tanto Naive Bayes como k-NN de scikit-learn pueden manejar variables en escalas diferentes. Sin embargo, esto es un punto de mejora potencial, especialmente para k-NN que es sensible a la escala de las variables.

El código para cargar y explorar el dataset incluye la configuración para evitar warnings en Windows relacionados con la detección de núcleos de CPU por parte de joblib:

```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.model_selection import StratifiedKFold, GridSearchCV, cross_val_predict
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Silenciar warnings de joblib en Windows
os.environ['LOKY_MAX_CPU_COUNT'] = '4'
warnings.filterwarnings('ignore', category=UserWarning, module='joblib')

# Cargar datos
df = pd.read_csv('data.csv')
print(df.info())
print(df['Bankrupt?'].value_counts())
print(f"\nPorcentaje de bancarrotas: {df['Bankrupt?'].mean() * 100:.2f}%")
```

3 Particionado y Validación Cruzada

He implementado una estrategia de validación cruzada estratificada de 5 folds, que es crucial por varias razones:

3.1 Estratificación

El particionado estratificado (`StratifiedKFold`) garantiza que cada fold mantenga la proporción original de clases (96.77 % / 3.23 %). Sin estratificación, algunos folds podrían tener muy pocas o ninguna instancia de bancarrota, haciendo imposible entrenar o evaluar adecuadamente los modelos.

Con 220 ejemplos positivos divididos en 5 folds, cada fold de test contendrá aproximadamente 44 bancarrotas, lo cual es suficiente para evaluar el rendimiento aunque sigue siendo un número bajo.

3.2 Importancia de cross_val_predict

Es fundamental usar `cross_val_predict` en lugar de entrenar sobre train y predecir sobre test manualmente, o peor aún, predecir sobre todo el dataset con un modelo entrenado en todo el dataset. Esta última práctica llevaría a métricas completamente optimistas y engañosas.

`cross_val_predict` garantiza que:

- Cada ejemplo se predice exactamente una vez
- La predicción se hace con un modelo que NO ha visto ese ejemplo durante entrenamiento
- Se obtienen predicciones para todo el dataset manteniendo la validez de la evaluación
- Las métricas calculadas son una estimación realista del rendimiento en datos nuevos

Esto es especialmente crítico en problemas con desbalance, donde evaluar incorrectamente puede ocultar problemas graves del modelo.

```
# Preparar datos
X = df.drop(columns=['Bankrupt?'])
y = df['Bankrupt?']

# Configurar validacion cruzada estratificada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

4 Clasificadores

He comparado los siguientes métodos usando validación cruzada correctamente:

- Naive Bayes Gaussiano
- Regla de k-NN (k-Nearest Neighbors)

4.1 Naive Bayes Gaussiano

He implementado un clasificador Naive Bayes Gaussiano, que asume que cada característica sigue una distribución normal y que todas las características son independientes entre sí dado la clase. Esta asunción de independencia ("naive") es claramente violada en datos financieros donde las variables están correlacionadas, pero el algoritmo suele funcionar razonablemente bien en la práctica.

4.1.1 Fundamento teórico

Naive Bayes aplica el teorema de Bayes con la asunción de independencia condicional:

$$P(y|x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Para cada clase y, estima $\mu_{y,i}$ y $\sigma^2_{y,i}$ para cada característica i, asumiendo:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma^2_{y,i}}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma^2_{y,i}}\right)$$

4.1.2 Implementación

Utilizo `cross_val_predict` para obtener predicciones robustas sobre todos los datos manteniendo la separación train/test:

```
# Entrenar Naive Bayes con validacion cruzada
nb = GaussianNB()
nb_preds = cross_val_predict(nb, X, y, cv=skf)

print(f"Accuracy: {accuracy_score(y, nb_preds):.4f}")
print(classification_report(y, nb_preds))
print(confusion_matrix(y, nb_preds))

# Entrenar modelo final con todos los datos
nb.fit(X, y)
```

4.2 k-Nearest Neighbors

He implementado el algoritmo k-NN (k-Nearest Neighbors), un método no paramétrico que clasifica cada ejemplo según la clase mayoritaria entre sus k vecinos más cercanos. A diferencia de Naive Bayes, k-NN no asume ninguna distribución particular de los datos.

4.2.1 Fundamento teórico

Para clasificar un punto $\mathbf{x}$, k-NN:

1. Calcula la distancia (típicamente Euclidiana) a todos los puntos del conjunto de entrenamiento
2. Selecciona los k puntos más cercanos
3. Asigna la clase mayoritaria entre esos k vecinos

La distancia Euclidiana entre dos puntos $\mathbf{x}$ y $\mathbf{x}'$ en $\mathbb{R}^{95}$ es:

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{95} (x_i - x_i')^2}$$

4.2.2 Consideraciones con desbalance

En problemas desbalanceados, k-NN puede tener dificultades porque:

- Los vecinos tienden a ser mayormente de la clase mayoritaria
- Un k muy grande diluye la señal de la clase minoritaria
- Un k muy pequeño aumenta la varianza y sensibilidad al ruido

4.2.3 Implementación

Sigo el mismo esquema de validación cruzada que en Naive Bayes:

```
# Evaluar con el mejor k encontrado
best_knn = KNeighborsClassifier(n_neighbors=k_optimo)
knn_preds = cross_val_predict(best_knn, X, y, cv=skf)

print(f"Accuracy: {accuracy_score(y, knn_preds):.4f}")
print(classification_report(y, knn_preds))
print(confusion_matrix(y, knn_preds))

# Entrenar modelo final con todos los datos
best_knn.fit(X, y)
```

5 Optimización de Hiperparámetros

He utilizado GridSearchCV para encontrar el valor óptimo de k en k-NN. La elección de k es crítica: valores pequeños capturan mejor patrones locales pero son sensibles al ruido, mientras que valores grandes dan predicciones más suaves pero pueden sobre-simplificar el problema.

5.1 Elección de métrica: F1-macro vs Accuracy

En problemas desbalanceados, la accuracy puede ser engañosa. Un clasificador que siempre predice "no bancarrota.ºbtendría 96.77 % de accuracy sin aportar valor alguno. Por ello, he usado `f1_macro` como métrica de optimización.

El F1-macro es la media aritmética del F1-score de cada clase:

$$\text{F1-macro} = \frac{\text{F1}_{\text{clase 0}} + \text{F1}_{\text{clase 1}}}{2}$$

Donde para cada clase:

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

Esta métrica trata ambas clases con igual importancia, forzando al modelo a equilibrar su rendimiento en lugar de ignorar la clase minoritaria.

5.2 Búsqueda en grid

He explorado valores de $k \in \{3, 5, 7, 9, 11, 15\}$, priorizando valores impares para evitar empates en votación binaria:

```
# Busqueda de hiperparametros con GridSearchCV
params_knn = {'n_neighbors': [3, 5, 7, 9, 11, 15]}
grid_knn = GridSearchCV(
    KNeighborsClassifier(),
    params_knn,
    cv=skf,
    scoring='f1_macro',
    n_jobs=-1
)
grid_knn.fit(X, y)

print(f"Mejor k encontrado: {grid_knn.best_params_['n_neighbors']}")
print(f"Mejor F1-score macro: {grid_knn.best_score_:.4f}")
```

6 Resultados

He recogido las métricas completas para ambos clasificadores usando validación cruzada. Los resultados revelan comportamientos muy diferentes entre los dos enfoques.

6.1 Resultados de Naive Bayes Gaussiano

- **Accuracy:** 0.2066 (20.66 %)
- **Precision clase 1 (bancarrota):** 0.03
- **Recall clase 1:** 0.85
- **F1-score clase 1:** 0.06
- **Matriz de confusión:** Detecta 187/220 bancarrotas (85 %) pero genera 5377 falsos positivos

Interpretación: Naive Bayes ha adoptado una estrategia extremadamente conservadora, clasificando la mayoría de ejemplos como bancarrota. Esto resulta en un recall excelente (85 % de detección) pero una precision catastrófica (solo 3 % de las predicciones positivas son correctas). El modelo es prácticamente inútil en producción porque generaría alarmas falsas constantemente.

6.2 Resultados de k-NN (k=3)

- **Accuracy:** 0.9658 (96.58 %)
- **Precision clase 1 (bancarrota):** 0.31
- **Recall clase 1:** 0.05
- **F1-score clase 1:** 0.09
- **F1-macro:** 0.5337 (mejor valor en grid search)
- **Matriz de confusión:** Detecta solo 11/220 bancarrotas (5 %) pero con 24 falsos positivos

Interpretación: k-NN muestra el problema opuesto: alta accuracy (96.58 %) pero esto es engañoso porque simplemente predice ”no bancarrotaçasi siempre. Solo detecta el 5 % de las bancarrotas reales. Sin embargo, cuando predice bancarrota, tiene un 31 % de probabilidad de estar en lo correcto (vs 3 % de Naive Bayes).

6.3 Comparación directa

```
# Comparacion final
print(f"Naive Bayes - Accuracy: {accuracy_score(y, nb_preds):.4f}")
print(f"KNN (k={grid_knn.best_params_['n_neighbors']}) - Accuracy: {accuracy_score(y,
    knn_preds):.4f}")
```

El trade-off es claro: Naive Bayes captura mejor la clase minoritaria (alto recall) a costa de precisión, mientras que k-NN prioriza precision general pero falla detectando bancarrotas.

Modelo	Accuracy	Recall (clase 1)	Precision (clase 1)
Naive Bayes	20.66 %	85 %	3 %
k-NN (k=3)	96.58 %	5 %	31 %

Cuadro 1: Comparación de rendimiento entre modelos

7 Discusión y Conclusiones

7.1 Análisis del impacto del desbalance

El desbalance extremo (30:1) domina completamente el comportamiento de ambos clasificadores. Este problema ilustra perfectamente por qué accuracy no es una métrica adecuada para datos desbalanceados:

- Un clasificador trivial que siempre predice "no bancarrota.obtendría 96.77 % accuracy
- Naive Bayes con 20.66 % accuracy es técnicamente "peor", pero detecta 85 % de bancarrotas
- k-NN con 96.58 % accuracy parece excelente, pero solo detecta 5 % de bancarrotas

7.2 Comportamientos distintivos de cada enfoque

7.2.1 Naive Bayes: Sesgo hacia la clase minoritaria

Naive Bayes, al modelar explícitamente las distribuciones de probabilidad de cada clase, parece haber encontrado que las características de bancarrota tienen distribuciones muy dispersas o solapadas con las no-bancarrotas. Esto lleva al modelo a ser excesivamente cauteloso, clasificando muchos casos normales como bancarrota.

Posibles causas:

- Violación severa de la asunción de independencia entre variables financieras
- Estimaciones pobres de μ y σ^2 para la clase minoritaria (solo 220 ejemplos)
- Distribuciones no-gaussianas en las características financieras

7.2.2 k-NN: Sesgo hacia la clase mayoritaria

k-NN sufre el problema opuesto. Con k=3, en la mayoría de regiones del espacio de características, los 3 vecinos más cercanos son empresas sanas simplemente porque hay 30 veces más ejemplos de esta clase. Las empresas en bancarrota quedan .ªisladas.º rodeadas de ejemplos negativos.

Incluso con k óptimo según F1-macro (k=3), el recall es solo 5 %. Valores más grandes de k empeoran aún más la situación al diluir completamente la señal de la clase minoritaria.

7.3 Importancia de la validación cruzada correcta

He implementado correctamente la validación cruzada usando `cross_val_predict`, lo cual ha sido fundamental para obtener estas conclusiones realistas. Evaluar incorrectamente (por ejemplo, entrenando y testeando sobre todo el dataset) habría ocultado estos problemas y dado una falsa sensación de confianza.

La diferencia es dramática: sin validación cruzada apropiada, los modelos mostrarían métricas artificialmente infladas que no se replicarían en producción con datos nuevos.

7.4 Mejoras futuras

Para abordar este problema adecuadamente, se requerirían técnicas especializadas para desbalance:

1. **Resampling:** SMOTE (Synthetic Minority Over-sampling), undersampling de la clase mayoritaria

2. **Pesos de clase:** Penalizar más los errores en la clase minoritaria durante entrenamiento
3. **Métodos ensemble:** Random Forest con balanceo, XGBoost con `scale_pos_weight`
4. **Normalización:** Especialmente crítico para k-NN que es sensible a escalas
5. **Selección de características:** Reducir dimensionalidad podría ayudar con 95 features
6. **Umbrales ajustados:** Modificar el threshold de decisión para balancear precision/recall

7.5 Conclusión final

Esta práctica demuestra que en presencia de desbalance severo:

- Los clasificadores básicos (Naive Bayes, k-NN) luchan significativamente
- La accuracy como métrica única es engañosa y peligrosa
- El F1-macro y el análisis detallado de matrices de confusión son esenciales
- La validación cruzada correcta es crítica para evitar conclusiones erróneas
- Se necesitan técnicas especializadas para problemas reales de este tipo

Ninguno de los dos modelos es actualmente viable para producción, pero han servido como baseline para entender el problema y la necesidad de enfoques más sofisticados.