Aplicación en Python del Modelo Credit Portfolio View para Microcréditos

Ángel Gustavo José Martínez

July 26, 2024

Abstract

Este documento expone la metodología para aplicar el modelo *Credit Portfolio View* (Wilson, 1997) a una cartera de crédito al consumo, considerando el impacto de variables macroeconómicas en la probabilidad de incumplimiento y su implementación en Python.
Se emplean modelos $ARIMA(2, d, q)$ para estimar los valores futuros de las variables macroeconómicas y simulaciones Monte Carlo para calcular la pérdida esperada y no esperada.
El propósito de este documento es proporcionar una referencia detallada para profesionales del sector bancario, específicamente aquellos dedicados a la gestión del riesgo de crédito al consumo.

Introducción

El modelo *Credit Portfolio View* emplea datos históricos de variables macroeconómicas y la probabilidad de incumplimiento (PI), junto con simulaciones basadas en el método de Monte Carlo, para estimar tanto las pérdidas esperadas (EL) como las pérdidas no esperadas (UL) de una cartera de créditos al consumo. No obstante, este enfoque es aplicable a cualquier tipo de cartera. Este modelo considera las fluctuaciones en las variables macroeconómicas que pueden influir en la probabilidad de incumplimiento.
El modelo está diseñado para ayudar a las instituciones financieras a evaluar el riesgo crediticio de sus carteras bajo distintos escenarios económicos. Asume que la probabilidad de incumplimiento depende de:

$$PI = \Phi(VE, \varepsilon, \nu)$$

Donde:

- VE: Comportamiento esperado de las variables macroeconómicas.
- ε: Errores en las predicciones de las variables macroeconómicas.
- ν: Sorpresas o choques macroeconómicos.

Este modelo ofrece múltiples ventajas, de entre las cuales destacan:

- **Incorporación de Factores Macroeconómicos:** Puede incorporar variables macroeconómicas como el PIB, la tasa de interés interbancaria, la inflación y el tipo de cambio. Esto permite una evaluación holística del riesgo de crédito , ya que estos factores pueden tener un impacto significativo en las probabilidades de incumplimiento.
- **Correlaciones entre deudores:** Modela correlaciones entre deudores a través de factores macroeconómicos, reflejando mejor el impacto de eventos económicos en la cartera.
- **Simulación de Escenarios:** Permite simular distintos escenarios económicos y evaluar su impacto, útil para pruebas de estrés y planificación de capital.
- **Flexibilidad y Escalabilidad:** Se adapta a diferentes tipos de carteras y puede manejar grandes volúmenes de datos.

En resumen, ofrece una evaluación más precisa y completa del riesgo de crédito al incluir factores macroeconómicos, modelar correlaciones entre deudores y ser flexible y escalable.

Metodología

En esta sección, se describe detalladamente la metodología para ajustar el modelo y su implementación en Python. Es fundamental subrayar que la implementación aquí presentada es meramente ilustrativa y ofrece una visión general de cómo podría ejecutarse. En un entorno más realista, se llevan a cabo diversos análisis preliminares:

- Análisis exploratorio exhaustivo de la información considerada para el modelo.
- Selección rigurosa de las variables macroeconómicas relevantes.
- Segmentación por producto, segmento de riesgo u otros criterios que pudieran hacer una diferencia en las probabilidades de incumplimiento.
- Validación de los supuestos y evaluación del desempeño de los modelos ajustados.
- Ejecución de pruebas de backtesting para asegurar la robustez del modelo.

0.1 Ajuste de Modelos ARIMA

Para cada variable macroeconómica X_i se ajusta un modelo $ARIMA(2, d, q)$ donde el orden de diferenciación (d) y el orden del componente de promedio móvil (q) se determinan a través de procedimientos de prueba y validación:

$$X_{i,t} = \phi_{i,1}X_{i,t-1} + \phi_{i,2}X_{i,t-2} + e_{i,t}$$

donde $\phi_{i,1}$ y $\phi_{i,2}$ son los parámetros del modelo ARIMA y $e_{i,t}$ son los errores o residuos del modelo.

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm

# Lectura del conjunto de datos
df = pd.read_csv('datos.csv')

# Definición de la variable objetivo
target = 'probabilidad'

# Nombres de las variables macroeconómicas
macro_vars = ['macro_var1', 'macro_var2']

# Parámetros del modelo ARIMA(2). Se eligen d=0 y q=0 por simplicidad.
arima_order = (2, 0, 0)

# Ajustar el modelo ARIMA para cada variable macroeconómica
macro_cols = list(set(macro_vars))
arima_models = {col: ARIMA(df[col], order=arima_order).fit() for col in
    macro_cols}
```

0.2 Ajuste de Regresión

Para cuantificar el impacto de las variables macroeconómicas en la probabilidad de incumplimiento, se ajusta un modelo de regresión en el que las variables independientes son los valores ajustados obtenidos a partir de los modelos $ARIMA(2, d, q)$, mientras que la variable dependiente es la transformación en $log - odds$ de P_1. Esta transformación se define como:

$$\text{log-odds}(p_i) = \log\left(\frac{p_i}{1 - p_i}\right).$$

Por otro lado, el modelo de regresión se expresa de la siguiente manera:

$$\text{log-odds}(p_i) = \beta_0 + \beta_1 \hat{X}_{1,t} + \beta_2 \hat{X}_{2,t} + \cdots + \beta_k \hat{X}_{k,t} + r_t,$$

donde $\hat{X}_{i,t}$ son los valores ajustados de las variables macroeconómicas y $\beta_0, \ldots, \beta_k$ son los coeficientes de regresión estimados, y r_t es el término de error en el tiempo t.

```python
# Obtener valores ajustados de las variables macroeconómicas utilizando los
    modelos ARIMA
macro_feats = pd.DataFrame({col: model.fittedvalues for col, model in
    arima_models.items()})

# Añadir una constante necesaria para la regresión lineal
macro_feats = sm.add_constant(macro_feats)

# Definir una función para la transformación logit de la variable objetivo
tf_odds = lambda pis: np.log(pis / (1 - pis))

# Aplicar la transformación logit a la variable objetivo
odds_target = tf_odds(df[target])

# Ajustar el modelo de regresión lineal usando los valores ajustados de las
    variables macroeconómicas
lm = sm.OLS(odds_target, macro_feats).fit()
```

0.3 Residuales y Covarianza

Obtenemos los residuos e_t de los modelos $ARIMA(2, d, q)$ ajustados y se calcula la matriz de covarianza Σ:

$$\Sigma = \text{Cov}(e_t).$$

```python
# Obtener residuales de los modelos ARIMA para cada variable macroeconómica
resids = pd.DataFrame({col: model.resid for col, model in arima_models.items
    ()})

# Insertar los residuales del modelo de regresión lineal en la primera
    columna
resids.insert(0, 'odds', lm.resid)

# Ajustar los residuales restando la media
resids = resids - resids.mean()

# Calcular la matriz de covarianza de los residuales ajustados
cov_matrix = resids.cov()
```

0.4 Generación de Choques Aleatorios

Se generan choques aleatorios (escenarios macroeconómicos) correlacionados usando la descomposición de Cholesky L de la matriz de covarianza Σ:

$$LL^T = \Sigma.$$

Los choques aleatorios Z_t se generan como:

$$Z_t \sim \mathcal{N}(0, I)$$

$$u_t = LZ_t.$$

```python
# Definir el número de trayectorias y el número de pasos
num_paths = 10000  # Número de trayectorias aleatorias a generar
steps = 1          # Num. de pasos en cada trayectoria (horizonte de riesgo)

# Obtener la descomposición de Cholesky de la matriz de covarianza (excluyendo la primera columna)
cholesky_decomp = np.linalg.cholesky(cov_matrix.iloc[1:, 1:])

# Generar choques aleatorios normales con la forma (num_paths, steps, número de variables macroeconómicas)
random_shocks = np.random.normal(size=(num_paths, steps, len(macro_cols)))

# Aplicar la descomposición de Cholesky para obtener choques correlacionados
correlated_shocks = random_shocks @ cholesky_decomp.T

# Transponer los choques para que las dimensiones sean (num_paths, número de variables macroeconómicas, steps)
correlated_shocks = correlated_shocks.transpose(0, 2, 1)
```

0.5 Simulación de Pronósticos Macroeconómicos y Probabilidades de Incumplimiento

Para cada camino de simulación y cada paso temporal (horizonte de riesgo), se actualizan las variables macroeconómicas y las probabilidades de incumplimiento:

$$\hat{X}_{i,t} = X_{i,t} + \epsilon_{i,t},$$

$$\text{log-odds}(p_i) = \beta_0 + \beta_1 \hat{X}_{1,t} + \beta_2 \hat{X}_{2,t} + \cdots + \beta_k \hat{X}_{k,t}.$$

Por último, se transforman los log-odds de vuelta a probabilidades:

$$p_i = \frac{\exp(\text{log-odds}(p_i))}{1 + \exp(\text{log-odds}(p_i))}.$$

```python
# Definición de paramétros en base a la cartera analizada
num_loans = 25000
ead_individual = 15000
pi_distributions = []

# Simular probabilidades de incumplimiento
for i in range(num_paths):
    macro_forecast = macro_feats.iloc[-1, 1:].values + correlated_shocks[i, :, 0]
    macro_forecast = np.insert(macro_forecast, 0, 1)  # Añadir constante
    odd_forecast = lm.predict(macro_forecast)
    pi_simulated = np.exp(odd_forecast) / (1 + np.exp(odd_forecast))
    pi_distributions.append(np.repeat(pi_simulated, num_loans))
```

0.6 Simulación de la Tasa de Pérdida en caso de Incumplimiento (LGD)

Para la simulación de la tasa de pérdida en caso de incumplimiento (LGD), se determina una distribución general que se empleará para la simulación Monte Carlo. La tasa de pérdida en caso de incumplimiento se modela como una variable aleatoria, denotada como LGD_i, con una distribución definida en función de los parámetros del análisis. Es importante señalar que la elección de esta distribución puede ser fija, según la decisión del responsable de implementar la metodología.

$$\text{LGD}_i \sim f(\cdot \mid \boldsymbol{\theta}), \quad \text{con} \quad \boldsymbol{\theta} \in \Theta.$$

```python
# Simular LGD (Loss Given Default). En este caso, se supone una distribución Normal con parámetros:
lgd_mean = 0.4
lgd_std = 0.1
lgd_distributions = [np.random.normal(lgd_mean, lgd_std, num_loans) for _ in range(num_paths)]
```

1 Cálculo de Pérdidas Esperadas (EL)

Para cada simulación, calculamos la pérdida esperada (EL) como:

$$\text{EL}_i = \text{EAD} \times p_i \times \text{LGD}_i \tag{1}$$

donde:

- EAD es la Exposición al Momento de Incumplimiento.
- p_i es la Probabilidad de Incumplimiento del escenario i.
- LGD_i es la Tasa de Pérdida en caso de Incumplimiento del escenario i.

Agrupamos las pérdidas esperadas por simulación de la cartera y calculamos las estadísticas de interés:

$$\text{EL}_{\text{total}} = \sum_{i=1}^{N} \text{EL}_i \tag{2}$$

Calculamos la media, desviación estándar y percentil 95 de las pérdidas esperadas:

$$\mu_{\text{EL}} = E[\text{EL}_{\text{total}}] \tag{3}$$

$$\sigma_{\text{EL}} = \sqrt{\text{Var}(\text{EL}_{\text{total}})} \tag{4}$$

$$\text{EL}_{95} = \text{Percentil}_{95}(\text{EL}_{\text{total}}) \tag{5}$$

```python
# Convertir a DataFrames
pi_distribution_df = pd.DataFrame(np.array(pi_distributions).flatten(), columns=['PI'])
lgd_distribution_df = pd.DataFrame(np.array(lgd_distributions).flatten(), columns=['LGD'])

# Calcular pérdida esperada (EL)
el = pi_distribution_df['PI'] * ead_individual * lgd_distribution_df['LGD']
el_total = el.values.reshape(num_paths, num_loans).sum(axis=1)

# Estadísticas
mean_loss_total = el_total.mean()
std_loss_total = el_total.std()
```

2 Pérdida No Esperada (UL)

La pérdida no esperada (UL) se calcula de dos maneras:

1. Usando el valor z para un nivel de confianza del 95

$$\text{UL} = z_{0.95} \times \sigma_{\text{EL}} \tag{6}$$

2. Usando el percentil 95

$$\mathrm{UL} = \mathrm{EL}_{95} - \mu_{\mathrm{EL}} \tag{7}$$

```
percentile_95_loss_total = np.percentile(el_total, 95)
confidence_level = 0.95
z_value = 1.645
ul_total = z_value * std_loss_total
ul_percentile_total = percentile_95_loss_total - mean_loss_total
```

3 Visualización

Finalmente, visualizamos la distribución de las pérdidas esperadas con un histograma y marcamos la media y el percentil 95

```
# Visualización de la distribución de pérdidas esperadas
sns.set_style('white')
plt.figure(figsize=(9, 5))
plt.grid(True, linestyle='--', color='gray', alpha=0.3)

sns.histplot(el_total, bins=100, kde=True, color='gray', alpha=0.5, line_kws
    ={'color': 'black', 'linewidth': 2})
plt.axvline(mean_loss_total, linewidth=2, label='Media', color='black',
    linestyle='--')
plt.axvline(percentile_95_loss_total, linewidth=2, label='Percentil 95%',
    color='red', linestyle='--')
plt.xlabel('Pérdida esperada simulada de la cartera', fontsize=12)
plt.ylabel('Frecuencia', fontsize=12)
plt.title('Distribución de la Pérdida Esperada simulada de la cartera',
    fontsize=14, color='black')
plt.legend()
plt.grid(True)
plt.show()
```

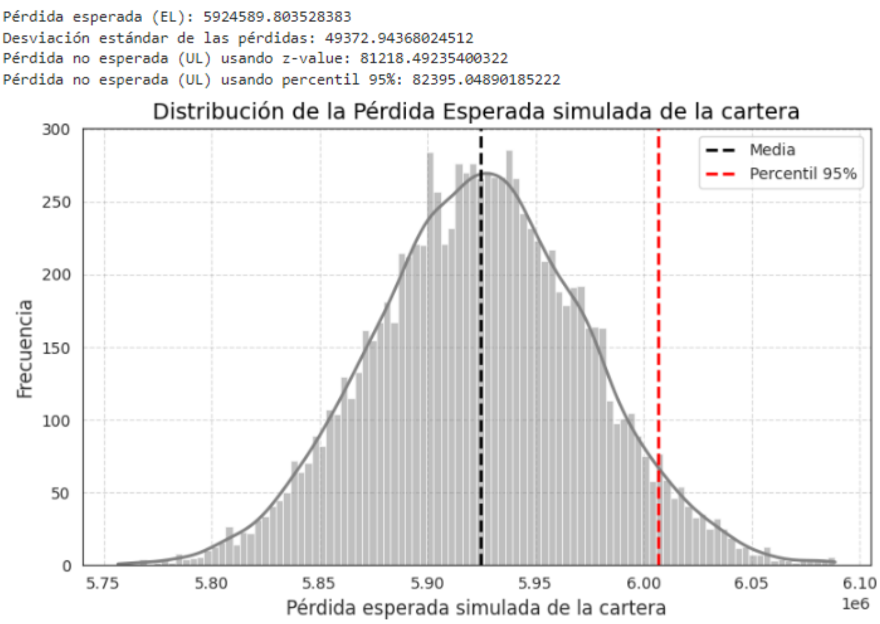

Figure 1: Distribución de la Pérdida Esperada para 10,000 simulaciones de una cartera de microcréditos.

4 Referencias

- Koulafetis, P. (2024). Modern credit risk management: Theory and practice. Palgrave Macmillan.
- Wilson, T. (1997a). Portfolio credit risk (I). *Risk Magazine*, 10(9).
- Wilson, T. (1997b). Portfolio credit risk (II). *Risk Magazine*, 10(10).