Universidad de Los Andes

Maestría en Inteligencia Analítica de Datos

Mini proyecto Deep Learning

Autores:

Cristian Mauricio Romero Buitrago

Jose Daniel Garcia Correa

Cesar Daniel Ramírez Cely

Departamento de Ingeniería Industrial

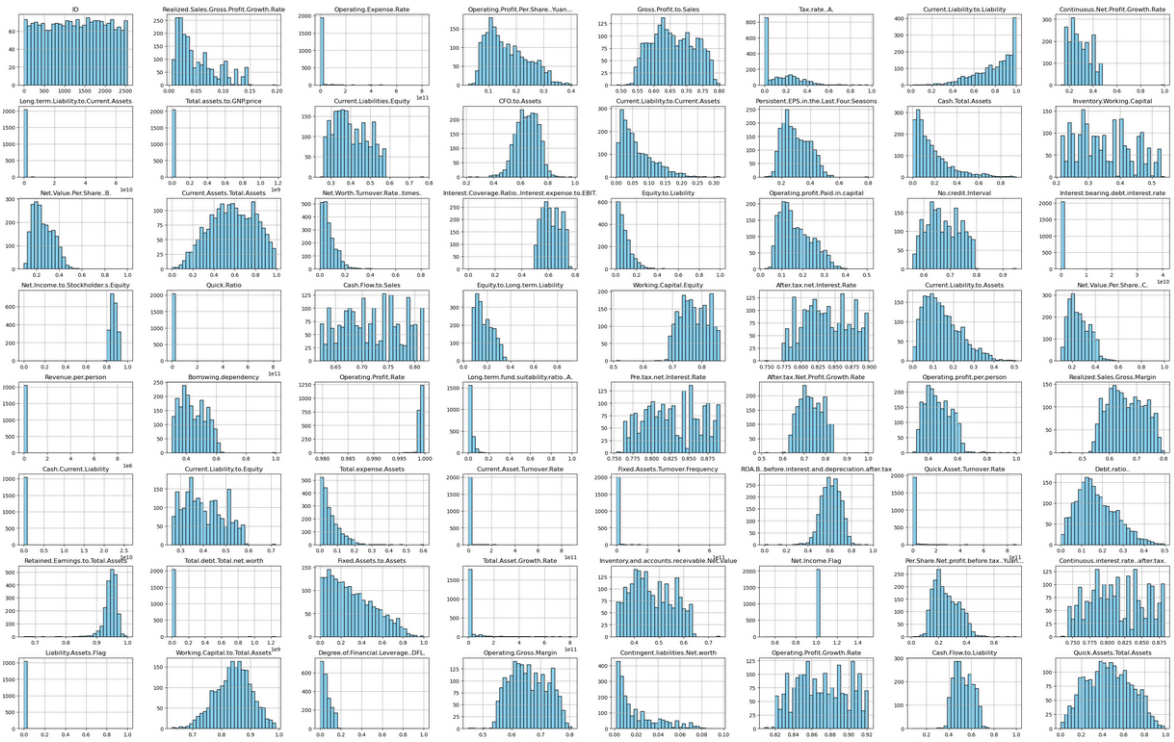
2025

1. Exploración de los datos para su entendimiento dentro del contexto organizacional.

   El conjunto de datos muestra una gran diversidad en los resultados financieros de las empresas, con varios casos extremos que sobresalen. Variables como el "Operating Profit Per Share" y el "Revenue per person" presentan medias mucho menores que sus desviaciones estándar, lo que indica que los datos están bastante dispersos y que existen valores atípicos marcados. Por otro lado, indicadores como el "Gross Profit to Sales" y el "Operating Expense Rate" muestran un comportamiento más uniforme, con medias y desviaciones estándar más equilibradas.

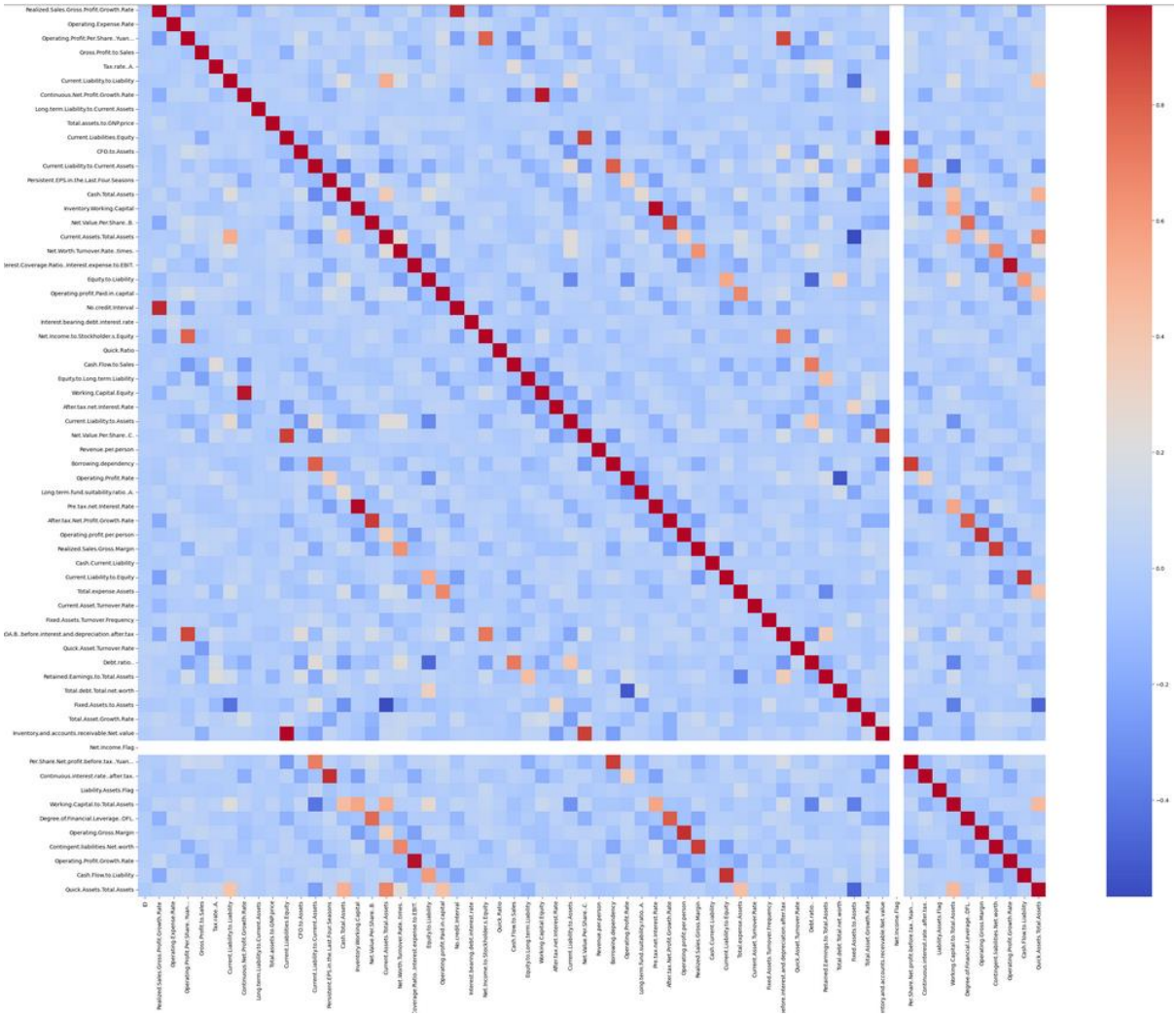
	ID	Realized.Sales.Gross.Profit.Growth.Rate	Operating.Expense.Rate	Operating.Profit.Per.Share..Yuan...	Gross.Profit.to.Sales	Tax.rate..A.
count	2050.00	2050.00	2.050000e+03	2050.00	2050.00	2050.00
mean	1276.68	0.05	1.143592e+10	0.17	0.66	0.15
std	733.77	0.04	6.637064e+10	0.07	0.07	0.17
min	1.00	0.01	0.000000e+00	0.03	0.49	0.00
25%	645.25	0.02	0.000000e+00	0.11	0.61	0.00
50%	1286.50	0.04	0.000000e+00	0.15	0.66	0.11
75%	1902.75	0.07	2.040574e+06	0.22	0.71	0.26
max	2550.00	0.20	8.153957e+11	0.40	0.80	0.97

   En general, se percibe que las razones financieras relacionadas con liquidez, rentabilidad y estructura presentan una alta variabilidad, algo esperable en un conjunto de empresas que se encuentran en diferentes etapas de crecimiento o que enfrentan distintos niveles de riesgo. También se detectan compañías con resultados financieros atípicos, ya que variables como el "Fixed Assets Turnover Frequency" y el "Current Asset Turnover Rate" alcanzan valores máximos muy elevados, lo que podría afectar el análisis si no se aplican métodos de escalamiento o transformación de datos.



Este conjunto de datos reúne 63 razones financieras extraídas de los estados financieros de empresas chinas que cotizaron en la Bolsa de Shanghái entre 1999 y 2009, con el propósito de predecir posibles bancarrotas. Al revisar cómo se distribuyen estas variables, se nota que muchas no siguen un patrón normal: presentan asimetrías, sesgos hacia la derecha y, en varios casos, valores extremos. Algunas razones financieras, como los indicadores de liquidez o endeudamiento, están naturalmente limitadas entre 0 y 1, mientras que otras, como los ingresos por persona o el efectivo sobre pasivos, muestran valores muy dispersos o exageradamente altos, lo que indica la necesidad de normalizar o transformar los datos antes de construir un modelo.

También es importante considerar que varias razones financieras miden aspectos similares como distintos tipos de rentabilidad o niveles de apalancamiento, lo que podría generar multicolinealidad entre las variables y afectar la estabilidad de los modelos si no se maneja bien. Dado que el objetivo es predecir un evento poco frecuente como la bancarrota, será clave asegurarse de que los datos estén balanceados.

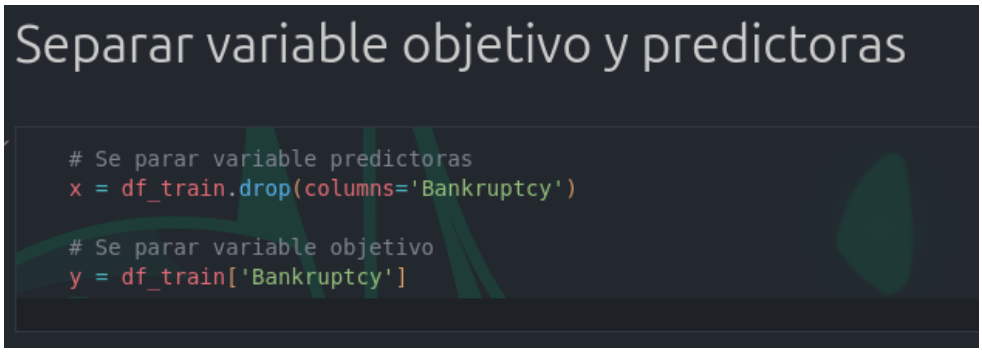


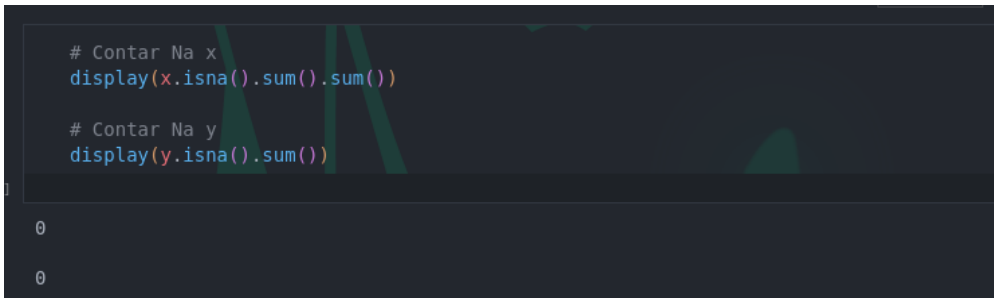
El análisis de la matriz de correlaciones muestra que, en general, las variables financieras no están fuertemente relacionadas entre sí, lo cual es positivo para su uso en modelos predictivos. Aun así, hay algunos grupos de indicadores de liquidez y rentabilidad que presentan alta correlación, lo que podría significar información redundante. También se detectaron correlaciones negativas moderadas, útiles para identificar relaciones inversas. Las zonas sin correlaciones podrían deberse a datos faltantes o diferencias entre variables. En conjunto, el panorama es favorable para avanzar, aunque será importante revisar las variables muy relacionadas antes de modelar.

2. Preparación de los datos para poder utilizarlos como entrada para modelos predictivos.

   Para preparar los datos de cara al modelado predictivo, se aplicaron técnicas de preprocesamiento siguiendo buenas prácticas. Primero se separan las columnas del data set inicial en variables predictoras “x” y variable objetivo “y”, Se realiza una validación de datos nulos o faltantes para ambos data sets, Se realiza la eliminación de

las variables predictoras que no son necesarias para la predicción tales como el id del registro el cual no aporta al modelo, se estandarizaron las variables con StandardScaler, logrando que todas tengan una media de cero y una varianza de uno, algo fundamental para modelos sensibles a la escala. Luego, para corregir el desbalance entre clases, se utilizó SMOTE, una técnica que genera datos sintéticos de la clase minoritaria hasta alcanzar aproximadamente el 80% del tamaño de la clase mayoritaria, permitiendo que el modelo aprenda de manera más equilibrada. Finalmente, los datos procesados se transformaron en arrays de tipo float32 para optimizar su uso en modelos de machine learning. Con estos pasos, los datos quedaron listos para iniciar el entrenamiento de modelos predictivos.





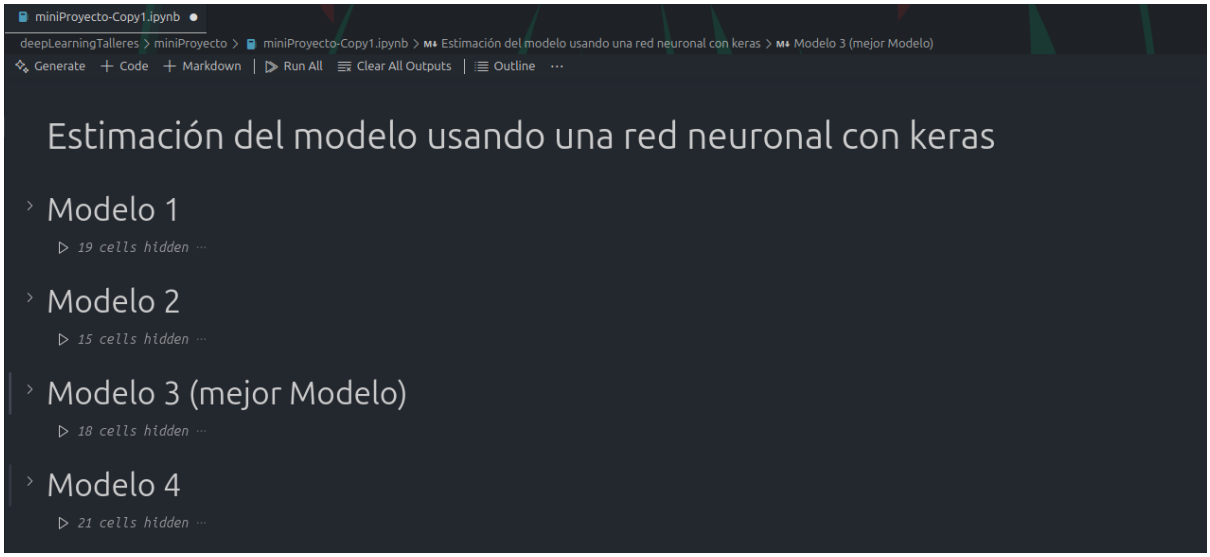
```python
# Se realiza el escalamiento de los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Manejar desbalanceo con SMOTE
sm = SMOTE(sampling_strategy=0.8, random_state=42)
X_resampled, y_resampled = sm.fit_resample(X_train_scaled, y_train)

# Transformar a formato NumPy
X_train_array = np.array(X_resampled, dtype=np.float32)
X_test_array = np.array(X_test_scaled, dtype=np.float32)

y_train_array = np.array(y_resampled, dtype=np.float32)
y_test_array = np.array(y_test, dtype=np.float32)
```

3. Análisis preliminar de selección de modelos relevantes para responder a la pregunta.

   Para la optimización de los hiperparámetros en un modelo de red neuronal utilizando Keras con técnicas como Dropout y EarlyStopping, es esencial definir una arquitectura adecuada utilizando el modelo secuencial de Keras. En primer lugar, debes apilar las capas necesarias para tu problema, como las capas densas (fully connected) y las capas de activación (como ReLU o Sigmoid), ajustando la cantidad de neuronas en cada capa según lo que mejor funcione para tu problema. El uso de la capa Dropout es crucial para prevenir el sobreajuste, ya que desconecta aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que permite al modelo generalizar mejor y no depender de características específicas. La técnica de EarlyStopping también juega un papel clave al detener el entrenamiento cuando el rendimiento en el conjunto de validación deja de mejorar, lo que ayuda a evitar el sobre entrenamiento y optimiza el uso de recursos computacionales. El algoritmo de optimización, como el gradiente descendente (Adam, SGD, o RMSprop), ajusta los pesos del modelo a través de las iteraciones de entrenamiento, y la calibración de los hiperparámetros, como la tasa de aprendizaje, puede mejorar significativamente el rendimiento del modelo.

   Una parte fundamental de este proceso es la calibración de los hiperparámetros, lo que se puede realizar utilizando técnicas como GridSearch. Estos métodos ayudan a encontrar la combinación óptima de parámetros, como la tasa de aprendizaje, el número de capas, la cantidad de neuronas por capa y la tasa de Dropout, entre otros. De manera alternativa, la optimización bayesiana también puede ser una opción avanzada para buscar de forma más eficiente en el espacio de hiperparámetros, en lugar de realizar una búsqueda exhaustiva. Tras cada entrenamiento, es crucial evaluar el modelo utilizando métricas adecuadas, como precisión, F1-score o MSE, dependiendo de si se trata de una tarea de clasificación o regresión. Con esta evaluación, puedes hacer ajustes finos en los hiperparámetros, repitiendo el proceso hasta obtener un

modelo optimizado que no solo tenga un buen rendimiento en el conjunto de entrenamiento, sino también en el conjunto de validación, asegurando su capacidad de generalización a datos no vistos.



4. Desarrollo y calibración de modelo

   Para el desarrollo y calibración del modelo se establecen cuatro modelos los cuales serán desarrollados y ejecutado para poder ver su rendimiento y seleccionar el mejor.

   Modelo 1

Entrenamiento modelo

```
model = keras.models.Sequential()

model.add(keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],), kernel_regularizer=regularizers.l2(1e-3)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

```
#opt=Adam(learning_rate=0.00001, beta_1=0.9, beta_2=0.999, epsilon=0.0000001)
opt=Adam(learning_rate=0.00001)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy', AUC(name='auc')])
```

```
early_stopping = EarlyStopping(
    monitor='val_auc',
    patience=12,
    mode='max',
    restore_best_weights=True
)
```

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss', factor=0.3, patience=4, min_lr=1e-6
)
```

```
history = model.fit(
    X_train_array, y_train_array,
    epochs=300,
    validation_split=0.15,
    callbacks=[early_stopping],
    class_weight=class_weights_dict,
    batch_size=32,
    verbose=1
)
```

Modelo 2

Entrenamiento del modelo

```
model2 = keras.models.Sequential()

model2.add(keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model2.add(keras.layers.BatchNormalization())
model2.add(keras.layers.Dropout(0.4))
model2.add(keras.layers.Dense(64, activation='relu'))
model2.add(keras.layers.BatchNormalization())
model2.add(keras.layers.Dropout(0.3))
model2.add(keras.layers.Dense(32, activation='relu'))
model2.add(keras.layers.BatchNormalization())
model2.add(keras.layers.Dropout(0.2))
model2.add(keras.layers.Dense(1, activation='sigmoid'))
```

```
opt2=Adam(learning_rate=0.0003)
opt=Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=0.0000001)
model2.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
```

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6
)
```

```
history2 = model2.fit(
    X_train_array, y_train_array,
    epochs=100,
    validation_split=0.15,
    callbacks=[early_stopping, reduce_lr],
    batch_size=64,
    verbose=2
)
```

Modelo 3 - campeón

Entrenamiento del modelo

```python
def entrenamiento_modelo(dim_entrada):
    model = keras.models.Sequential()

    model.add(keras.layers.Dense(256, activation='relu', input_shape=(dim_entrada,)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.5))

    model.add(keras.layers.Dense(128, activation='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.4))

    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.3))

    model.add(keras.layers.Dense(32, activation='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.2))

    model.add(keras.layers.Dense(1, activation='sigmoid'))

    opt = Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy', AUC(name='auc')])

    return model
```

```python
early_stopping = EarlyStopping(
    monitor='val_auc',
    patience=12,
    mode='max',
    restore_best_weights=True,
    verbose=1
)
```

```python
for fold, (train_idx, val_idx) in enumerate(skf.split(X_train_array, y_train_array)):
    print(f"\n--- Fold {fold + 1} ---")

    X_fold_train, X_fold_val = X_train_array[train_idx], X_train_array[val_idx]
    y_fold_train, y_fold_val = y_train_array[train_idx], y_train_array[val_idx]

    model = entrenamiento_modelo(X_fold_train.shape[1])

    history = model.fit(
        X_fold_train, y_fold_train,
        validation_data=(X_fold_val, y_fold_val),
        epochs=200,
        batch_size=32,
        class_weight=class_weights_dict,
        callbacks=[early_stopping],
        verbose=1)
```

Modelo 4

Entrenamiento del modelo

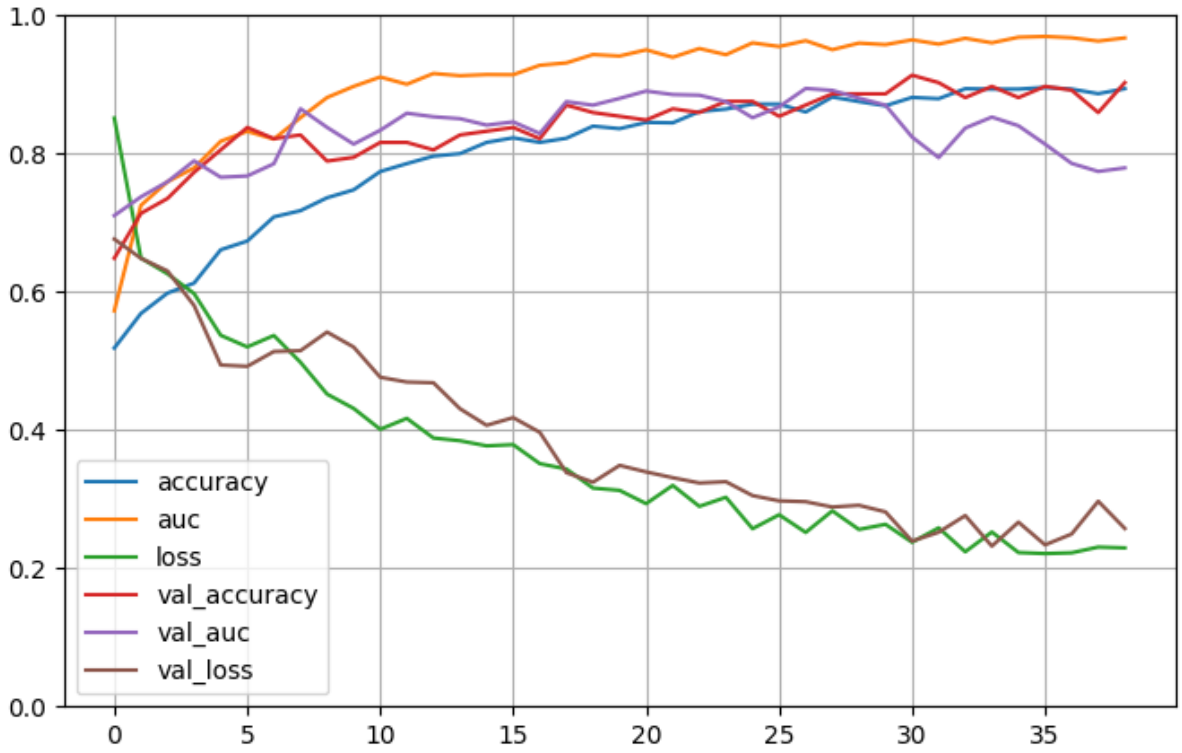
```python
def entrenamiento_modelo_2(dim_entrada, lr=5e-4, l2=1e-4, dropout=0.3):

    model = keras.models.Sequential()
    model.add(keras.layers.Dense(128, activation='selu', kernel_regularizer=regularizers.l2(l2), input_shape=(dim_entrada,)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(dropout))

    model.add(keras.layers.Dense(64, activation='selu', kernel_regularizer=regularizers.l2(l2)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(dropout))

    model.add(keras.layers.Dense(32, activation='selu', kernel_regularizer=regularizers.l2(l2)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(dropout/2))

    model.add(keras.layers.Dense(1, activation='sigmoid'))

    opt  = Adam(learning_rate=lr)
    #loss = BinaryFocalCrossentropy(alpha=0.75, gamma=2.0)
    loss = 'binary_crossentropy'
    model.compile(optimizer=opt, loss=loss, metrics=['accuracy', AUC(name='auc')])

    return model
```

```python
early_stopping = EarlyStopping(
    monitor='val_auc',
    patience=15,
    mode='max',
    restore_best_weights=True,
    verbose=1
)
```

```python
for fold, (train_idx, val_idx) in enumerate(skf.split(X_train_array, y_train_array)):
    print(f"\n--- Fold {fold + 1} ---")

    X_fold_train, X_fold_val = X_train_array[train_idx], X_train_array[val_idx]
    y_fold_train, y_fold_val = y_train_array[train_idx], y_train_array[val_idx]

    #X_tr_bal, y_tr_bal = smote.fit_resample(X_fold_train, y_fold_train)

    model = entrenamiento_modelo_2(dim_entrada=X_fold_train.shape[1])

    history = model.fit(
        X_fold_train, y_fold_train,
        validation_data=(X_fold_val, y_fold_val),
        epochs=200,
        batch_size=32,
        class_weight=class_weights_dict,
        callbacks=[early_stopping],
        verbose=1)
```

5. Visualización de resultados

   Se realiza el entrenamiento de todos los modelos los cuales generan predicciones al data set de la competencia y generan diferentes rendimientos. En cada iteración se generaron cambios a los modelos o la implementación de nuevos modelos. El modelo con el mejor rendimiento fue el modelo 4 el cual tiene la siguiente grafica de entrenamiento.



Con el modelo ya entrenado se genera la predicción con el formato solicitado por la competencia

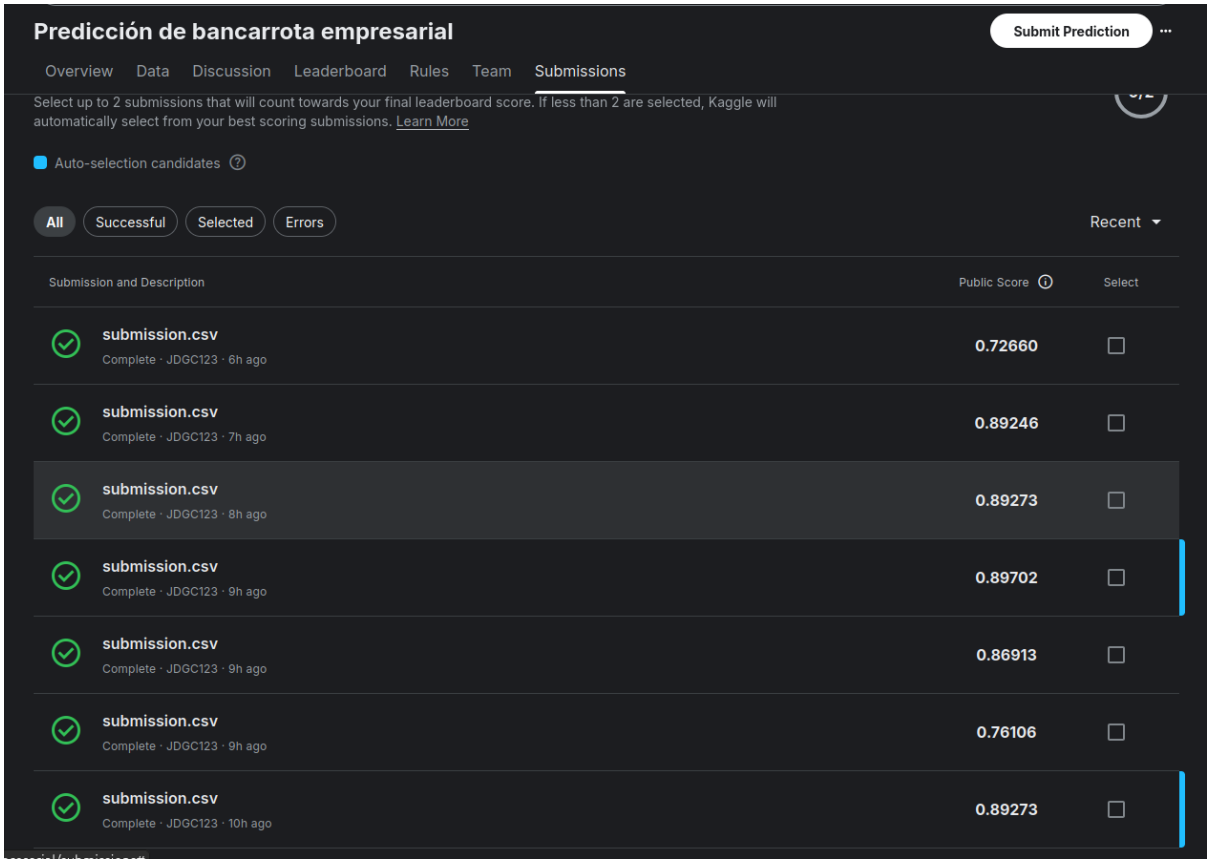
```
y_predict = [x for x in submission_preds]
```
✓ 0.0s

```
resultado = pd.DataFrame({'ID': df_test['ID'].values, 'Bankruptcy': y_predict})
resultado.head()
```
✓ 0.0s

	ID	Bankruptcy
0	5	0.355649
1	14	0.396247
2	16	0.091037
3	26	0.303608
4	28	0.082232

```
resultado.to_csv('submission.csv', index=False)
```
✓ 0.0s

Se generaron bastantes entradas en la competencia con cada modelo y cambio implementado



Nota: El modelo campeón y los últimos generados no tienen una sección de prueba debido a que para un mejor entrenamiento el set de train solo se divide para el entrenamiento y no para pruebas