ing-club-loan-default-prediction-1

July 16, 2025

1 Lending Club Loan Default Prediction

This notebook builds a deep learning model to predict whether a loan will be defaulted, using historical data from Lending Club (2007–2015).

[4]:
```
# Data handling & visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn tools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# TensorFlow & Keras (modern, clean style)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.utils import plot_model
```

1.1 Load Dataset

[6]:
```
# Replace with actual file path
data = pd.read_csv("loan_data.csv")
print(data.columns)

data.head()
```

```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

[6]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	\
0	1	debt_consolidation	0.1189	829.10	11.350407	
1	1	credit_card	0.1071	228.22	11.082143	

```
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

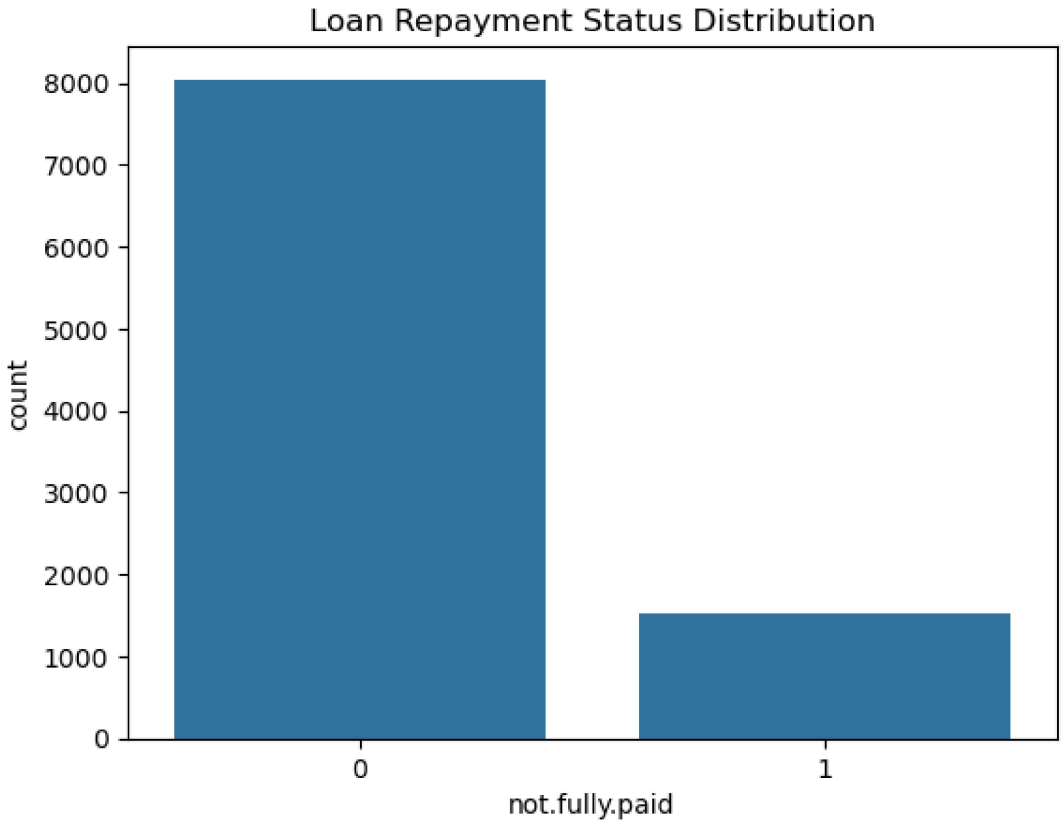
1.2 Data Preprocessing and Feature Transformation

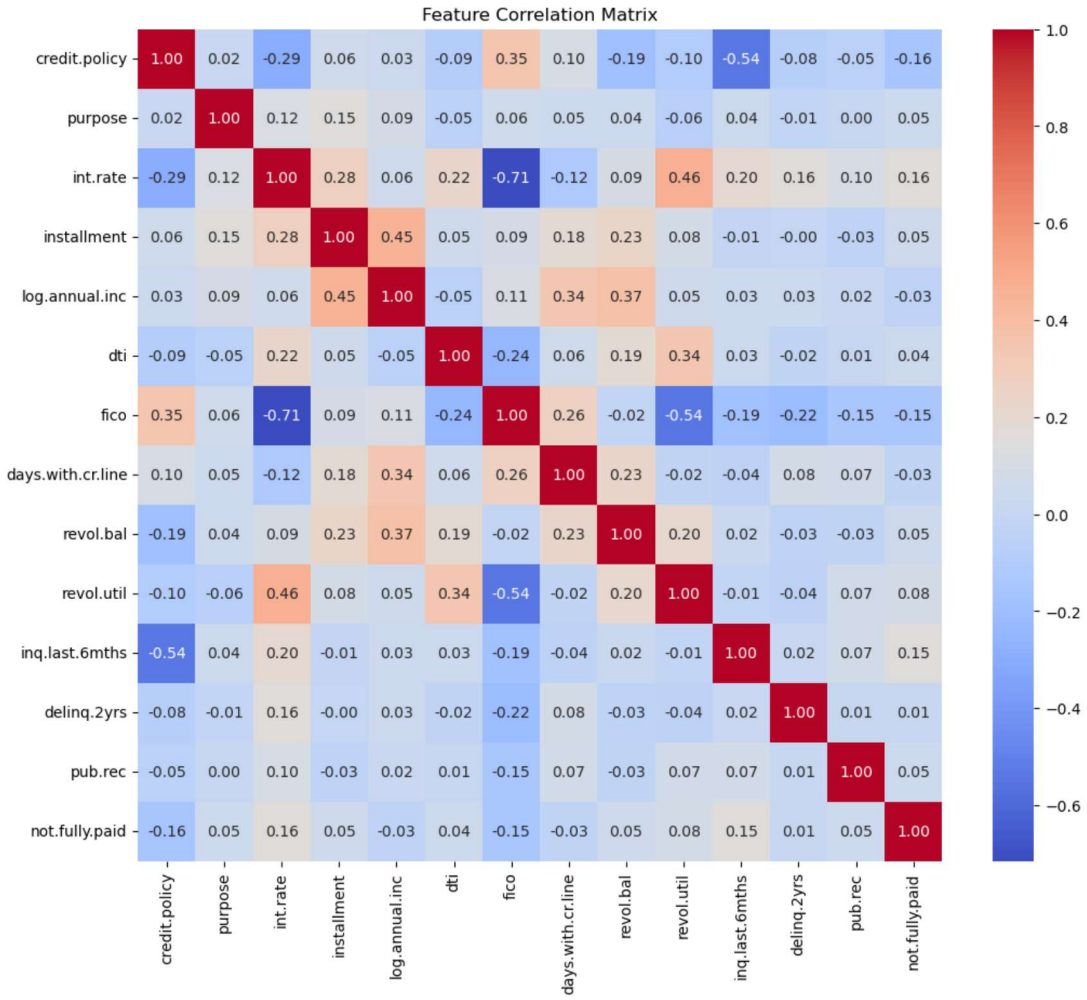
[8]:
```
# Encode categorical variable
data['purpose'] = LabelEncoder().fit_transform(data['purpose'])

# Check for missing values
print(data.isnull().sum())

# Fill or drop missing values
data = data.dropna()
```

```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
revol.bal            0
revol.util           0
inq.last.6mths       0
delinq.2yrs          0
pub.rec              0
not.fully.paid       0
dtype: int64
```

1.3 Exploratory Data Analysis

[10]:
```
# Distribution of target variable
sns.countplot(x='not.fully.paid', data=data)
plt.title("Loan Repayment Status Distribution")
plt.show()

# Correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()
```





1.4 Feature Engineering

[12]:
```
# Drop highly correlated features (example: remove one of two highly correlated features)
# Let's suppose we find 'installment' and 'int.rate' are highly correlated
data = data.drop(['installment'], axis=1)
```

1.5 Split and Scale Data

[14]:
```
X = data.drop('not.fully.paid', axis=1)
y = data['not.fully.paid']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1.6 Deep Learning Model

[16]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense

model = Sequential([
    Input(shape=(X_train.shape[1],)),       #  Explicit input layer
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')          # Use 'softmax' for multiclass
])

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/20
192/192             4s 9ms/step -
accuracy: 0.8359 - loss: 0.4844 - val_accuracy: 0.8415 - val_loss: 0.4257
Epoch 2/20
192/192             1s 6ms/step -
accuracy: 0.8354 - loss: 0.4279 - val_accuracy: 0.8415 - val_loss: 0.4243
Epoch 3/20
192/192             1s 5ms/step -
accuracy: 0.8398 - loss: 0.4148 - val_accuracy: 0.8415 - val_loss: 0.4196
Epoch 4/20
192/192             1s 5ms/step -
accuracy: 0.8484 - loss: 0.3995 - val_accuracy: 0.8421 - val_loss: 0.4193
Epoch 5/20
192/192             1s 5ms/step -
accuracy: 0.8449 - loss: 0.4039 - val_accuracy: 0.8421 - val_loss: 0.4168
Epoch 6/20
192/192             1s 4ms/step -
accuracy: 0.8483 - loss: 0.4032 - val_accuracy: 0.8447 - val_loss: 0.4163
Epoch 7/20
192/192             1s 5ms/step -
accuracy: 0.8419 - loss: 0.4111 - val_accuracy: 0.8447 - val_loss: 0.4157
Epoch 8/20
192/192             1s 6ms/step -
accuracy: 0.8294 - loss: 0.4221 - val_accuracy: 0.8421 - val_loss: 0.4147
Epoch 9/20
```

```
192/192             1s 6ms/step -
accuracy: 0.8389 - loss: 0.4109 - val_accuracy: 0.8434 - val_loss: 0.4145
Epoch 10/20
192/192             1s 6ms/step -
accuracy: 0.8412 - loss: 0.4093 - val_accuracy: 0.8434 - val_loss: 0.4159
Epoch 11/20
192/192             1s 6ms/step -
accuracy: 0.8426 - loss: 0.4017 - val_accuracy: 0.8434 - val_loss: 0.4134
Epoch 12/20
192/192             1s 5ms/step -
accuracy: 0.8454 - loss: 0.3981 - val_accuracy: 0.8441 - val_loss: 0.4150
Epoch 13/20
192/192             1s 5ms/step -
accuracy: 0.8372 - loss: 0.4036 - val_accuracy: 0.8428 - val_loss: 0.4141
Epoch 14/20
192/192             1s 5ms/step -
accuracy: 0.8388 - loss: 0.4079 - val_accuracy: 0.8428 - val_loss: 0.4133
Epoch 15/20
192/192             1s 5ms/step -
accuracy: 0.8393 - loss: 0.4029 - val_accuracy: 0.8441 - val_loss: 0.4135
Epoch 16/20
192/192             1s 5ms/step -
accuracy: 0.8448 - loss: 0.3941 - val_accuracy: 0.8454 - val_loss: 0.4139
Epoch 17/20
192/192             1s 5ms/step -
accuracy: 0.8426 - loss: 0.3983 - val_accuracy: 0.8447 - val_loss: 0.4131
Epoch 18/20
192/192             1s 5ms/step -
accuracy: 0.8443 - loss: 0.3932 - val_accuracy: 0.8421 - val_loss: 0.4125
Epoch 19/20
192/192             1s 5ms/step -
accuracy: 0.8412 - loss: 0.3974 - val_accuracy: 0.8428 - val_loss: 0.4142
Epoch 20/20
192/192             1s 5ms/step -
accuracy: 0.8428 - loss: 0.3930 - val_accuracy: 0.8428 - val_loss: 0.4126
```

1.7 Model Evaluation

[18]:
```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
60/60             0s 3ms/step -
accuracy: 0.8491 - loss: 0.3993
Test Accuracy: 0.84
```

```
60/60             0s 4ms/step
[[1606    5]
 [ 302    3]]
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      1611
           1       0.38      0.01      0.02       305

    accuracy                           0.84      1916
   macro avg       0.61      0.50      0.47      1916
weighted avg       0.77      0.84      0.77      1916
```