NB_6_custom_Loss

November 22, 2025

[9]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪accuracy_score
import torch
import torch.nn as nn
import torch.nn.functional as F

# Set random seeds for reproducibility
torch.manual_seed(0)
np.random.seed(0)

print("Imports OK  ")
```

```
Imports OK
```

[10]:
```python
# 1. Load Data
filename = "/Users/gasper/Documents/PSL/year_1/semster_1/
 ↪Pratical-Machine-Learning/NB6 - Gradient Descent/Fraud_detection.csv"
df = pd.read_csv(filename)
df_clean = df.dropna()

print(f"Dataset shape: {df_clean.shape}")

# 2. Identify Amount Column
# We need to extract the Amount BEFORE splitting to ensure we don't lose track␣
 ↪of it
if "Amount" in df_clean.columns:
    AMOUNT_COL = "Amount"
elif "amount" in df_clean.columns:
    AMOUNT_COL = "amount"
else:
    # Fallback: try to find the first numeric column that isn't the target
    cols = df_clean.select_dtypes(include=[np.number]).columns.tolist()
    if "target" in cols:
        cols.remove("target")
    AMOUNT_COL = cols[0]

print(f"Using '{AMOUNT_COL}' as the transaction amount column.")

# 3. Split Data (X, y, and Amounts)
# We split X and y as usual, but we also need to split the Amounts array
# so it matches the X/y indices exactly.
X = df_clean.drop(columns=["target"]).values
y = df_clean["target"].values
amounts = df_clean[AMOUNT_COL].values

# Train/Temp Split
X_tr, X_tmp, y_tr, y_tmp, amt_tr, amt_tmp = train_test_split(
    X, y, amounts, test_size=0.4, random_state=42, stratify=y
)

# Val/Test Split
X_va, X_te, y_va, y_te, amt_va, amt_te = train_test_split(
    X_tmp, y_tmp, amt_tmp, test_size=0.5, random_state=43, stratify=y_tmp
)

# 4. Convert to PyTorch Tensors
# Note: We need amounts as float32 for the loss calculation
X_train_t = torch.tensor(X_tr, dtype=torch.float32)
y_train_t = torch.tensor(y_tr, dtype=torch.float32)  # Float needed for custom␣
 ↪loss
amt_train_t = torch.tensor(amt_tr, dtype=torch.float32)

X_val_t = torch.tensor(X_va, dtype=torch.float32)
y_val_t = torch.tensor(y_va, dtype=torch.float32)
amt_val_t = torch.tensor(amt_va, dtype=torch.float32)

X_test_t = torch.tensor(X_te, dtype=torch.float32)
y_test_t = torch.tensor(y_te, dtype=torch.float32)
amt_test_t = torch.tensor(amt_te, dtype=torch.float32)

print(f"Train set: {X_train_t.shape}")
print(f"Val set:   {X_val_t.shape}")
print(f"Test set:  {X_test_t.shape}")
```

```
Dataset shape: (164492, 30)
Using 'Amount' as the transaction amount column.
Train set: torch.Size([98695, 29])
Val set:   torch.Size([32898, 29])
Test set:  torch.Size([32899, 29])
```

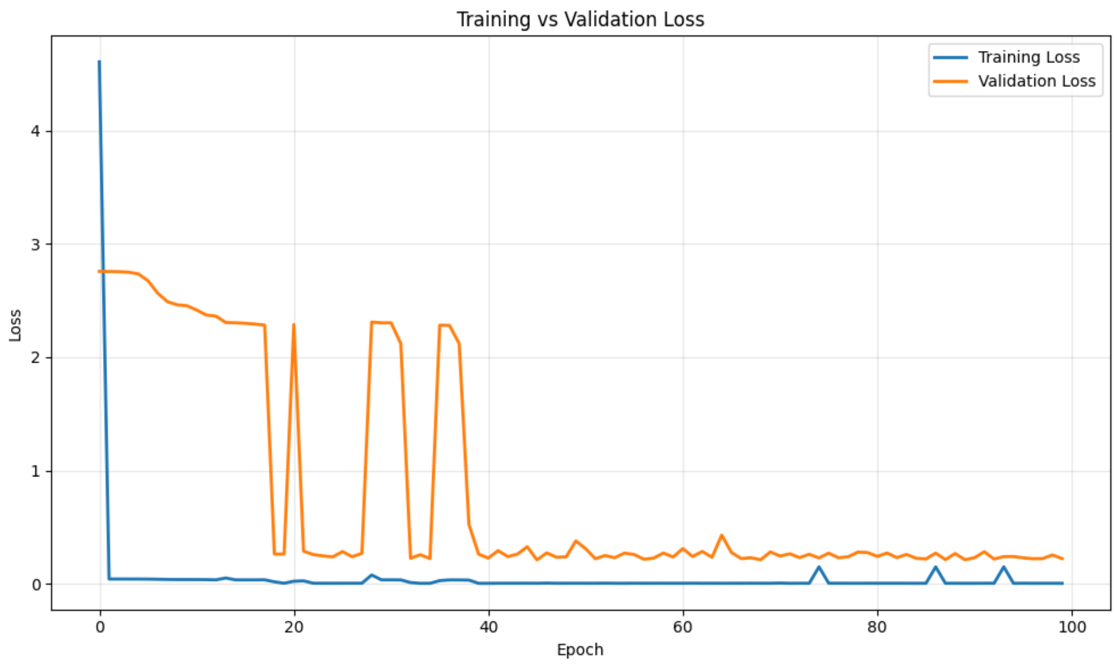
```
[11]: class ExpectedCostLoss(nn.Module):
          """
          Custom loss function that minimizes total business cost.
          """

          def __init__(
              self, abandonment_limit=0.1, user_lazy_factor=0.2,␣
       ↪penalty_per_percent=10000
          ):
              super().__init__()
              self.limit = abandonment_limit
              self.lazy_factor = user_lazy_factor
              # 10,000 EUR penalty for every 1% (0.01) means we multiply by 1,000,000
              # Example: 0.11 - 0.10 = 0.01 * 1,000,000 = 10,000
              self.penalty_scaler = penalty_per_percent * 100

          def forward(self, logits, targets, amounts):
              """
              logits:  (batch_size, 1) Raw outputs
              targets: (batch_size, )  0 for Legit, 1 for Fraud
              amounts: (batch_size, )  Transaction values in EUR
              """
              # 1. Get Probabilities
              probs = torch.sigmoid(logits).squeeze()

              # 2. Authentication Cost
              # We pay ~1 EUR for every high probability assignment
              auth_cost = probs * 1.0

              # 3. Missed Fraud Cost
              # If target=1 and prob=0.1, we miss 0.9 * Amount
              missed_fraud_cost = targets * (1 - probs) * amounts

              # 4. Abandonment Penalty (Global Batch Constraint)
              avg_auth_rate = probs.mean()
              current_abandon_rate = avg_auth_rate * self.lazy_factor

              # ReLU handles the "cliff": if rate < limit, penalty is 0
              excess_abandonment = F.relu(current_abandon_rate - self.limit)
              abandonment_penalty = excess_abandonment * self.penalty_scaler

              # 5. Total Loss
              # We average per-transaction costs and add the global penalty
              total_loss = (auth_cost + missed_fraud_cost).mean() +␣
       ↪abandonment_penalty

              return total_loss
```

[12]:
```python
class CostOptimizedNN(nn.Module):
    def __init__(self, input_dim, hidden_dim=32):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 1),  # Output 1 scalar (logit)
        )

    def forward(self, x):
        return self.net(x)


# Initialize
model = CostOptimizedNN(input_dim=X_train_t.shape[1])
criterion = ExpectedCostLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 64
N = X_train_t.shape[0]

# Track losses
train_losses = []
val_losses = []

print("Starting Cost-Sensitive Training...")

for epoch in range(epochs):
    model.train()
    perm = torch.randperm(N)
    total_loss = 0.0

    for start in range(0, N, batch_size):
        idx = perm[start : start + batch_size]
        xb = X_train_t[idx]
        yb = y_train_t[idx]
        ab = amt_train_t[idx]  # Batch amounts

        # Forward
        logits = model(xb)
        loss = criterion(logits, yb, ab)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # Calculate average training loss for the epoch
    avg_train_loss = total_loss / N
    train_losses.append(avg_train_loss)

    # Validation Check
    model.eval()
    with torch.no_grad():
        val_logits = model(X_val_t)
        val_loss = criterion(val_logits, y_val_t, amt_val_t).item()
        val_losses.append(val_loss)

    if epoch % 10 == 0:
        print(
            f"Epoch {epoch}: Train Loss: {avg_train_loss:.4f} | Val Cost Score:␣
 ↪{val_loss:.2f}"
        )

print("Training Complete.")
```

```
Starting Cost-Sensitive Training…
Epoch 0: Train Loss: 4.6067 | Val Cost Score: 2.76
Epoch 10: Train Loss: 0.0365 | Val Cost Score: 2.42
Epoch 20: Train Loss: 0.0218 | Val Cost Score: 2.29
Epoch 30: Train Loss: 0.0341 | Val Cost Score: 2.30
Epoch 40: Train Loss: 0.0034 | Val Cost Score: 0.23
Epoch 50: Train Loss: 0.0039 | Val Cost Score: 0.31
Epoch 60: Train Loss: 0.0039 | Val Cost Score: 0.31
Epoch 70: Train Loss: 0.0057 | Val Cost Score: 0.24
Epoch 80: Train Loss: 0.0038 | Val Cost Score: 0.24
Epoch 90: Train Loss: 0.0036 | Val Cost Score: 0.23
Training Complete.
```

```
[ ]: # Plot Training and Validation Loss
plt.figure(figsize=(10, 6))

plt.plot(train_losses, label="Training Loss", linewidth=2)
plt.plot(val_losses, label="Validation Loss", linewidth=2)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
```

```python
plt.show()

print(f"\nFinal Training Loss: {train_losses[-1]:.4f}")
print(f"Final Validation Loss: {val_losses[-1]:.2f}")
```



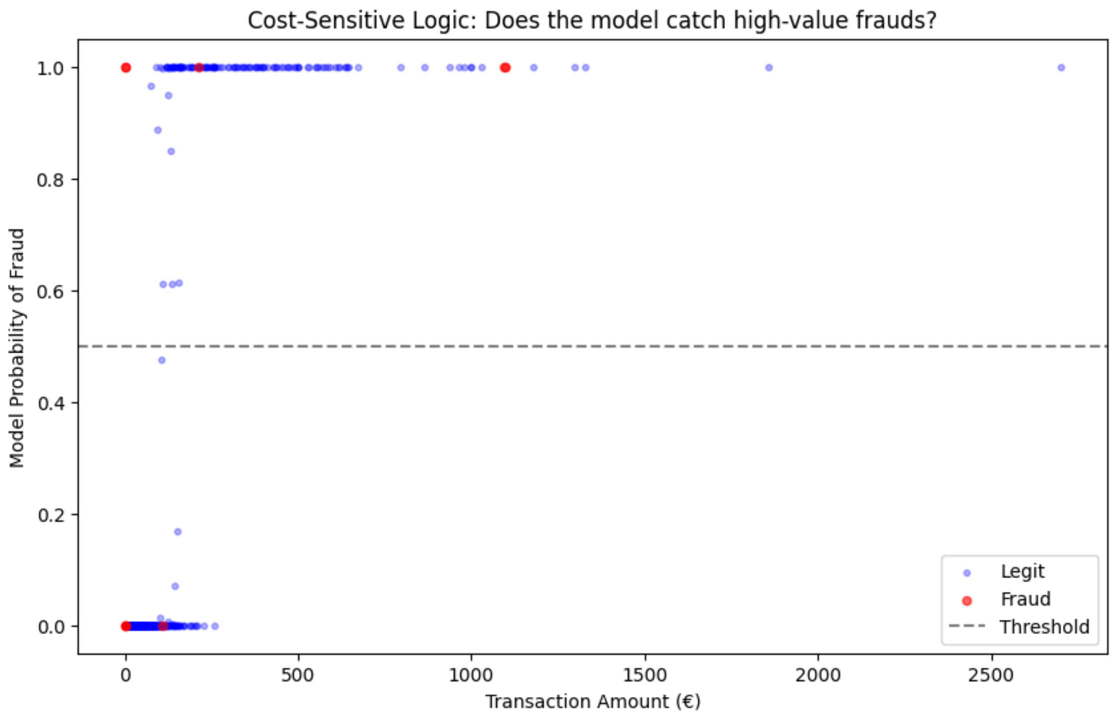
```
Final Training Loss: 0.0034
Final Validation Loss: 0.22
```

[18]:
```python
def calculate_bank_invoice(model, X_tensor, y_true, amounts, threshold=0.5):
    model.eval()
    with torch.no_grad():
        logits = model(X_tensor)
        probs = torch.sigmoid(logits).squeeze().numpy()

    # Make hard decisions based on threshold
    preds = (probs >= threshold).astype(int)

    total_transactions = len(y_true)

    # 1. Authentication Costs
    auth_count = (preds == 1).sum()
    auth_fee = auth_count * 1.0

    # 2. Missed Fraud Costs
    # We let it go (pred=0) but it was fraud (y=1)
    missed_indices = np.where((preds == 0) & (y_true == 1))[0]
    missed_fraud_cost = amounts[missed_indices].sum()

    # 3. Abandonment Penalty
    abandon_rate = (auth_count * 0.2) / total_transactions
    excess = max(0, abandon_rate - 0.10)
    # 10k for every 1% (0.01)
    abandon_penalty = 10000 * (excess * 100)

    total_cost = auth_fee + missed_fraud_cost + abandon_penalty

    print(f"--- FINAL BANK INVOICE ---")
    print(f"Auth Requested:     {auth_count} (€{auth_fee:.2f})")
    print(f"Missed Fraud Cost:  €{missed_fraud_cost:,.2f}")
    print(f"Abandonment Rate:   {abandon_rate:.2%} (Penalty: €{abandon_penalty:
↪,.2f})")
    print(f"--------------------------")
    print(f"TOTAL COST:         €{total_cost:,.2f}")

    return total_cost, preds


# Run Evaluation on Test Set
total_cost, final_preds = calculate_bank_invoice(
    model,
    X_test_t,
    y_te,  # Use the numpy array for y_true
    amt_te,  # Use the numpy array for amounts
    threshold=0.5,
)
```

```
--- FINAL BANK INVOICE ---
Auth Requested:     4511 (€4511.00)
Missed Fraud Cost:  €2,882.21
Abandonment Rate:   2.74% (Penalty: €0.00)
--------------------------
TOTAL COST:         €7,393.21
```

[19]:
```
# Get probabilities for plotting
with torch.no_grad():
    probs = torch.sigmoid(model(X_test_t)).squeeze().numpy()

plt.figure(figsize=(10, 6))

# Scatter plot: X-axis = Amount, Y-axis = Predicted Probability
# Color = Actual Class (Red=Fraud, Blue=Legit)
```

```python
subset = np.random.choice(len(amt_te), size=min(1000, len(amt_te)),␣
 ↪replace=False)

plt.scatter(
    amt_te[subset][y_te[subset] == 0],
    probs[subset][y_te[subset] == 0],
    alpha=0.3,
    label="Legit",
    color="blue",
    s=10,
)
plt.scatter(
    amt_te[subset][y_te[subset] == 1],
    probs[subset][y_te[subset] == 1],
    alpha=0.6,
    label="Fraud",
    color="red",
    s=20,
)

plt.axhline(0.5, color="gray", linestyle="--", label="Threshold")
plt.xlabel("Transaction Amount (€)")
plt.ylabel("Model Probability of Fraud")
plt.title("Cost-Sensitive Logic: Does the model catch high-value frauds?")
plt.legend()
plt.show()
```


Cost-Sensitive Logic: Does the model catch high-value frauds?
Model Probability of Fraud
Transaction Amount (€)
Legit
Fraud
Threshold