Supplementary Material: Source codes for different

models Classical Machine Learning Models (Structured

Data) Models evaluated:
- Random Forest
- Gradient Boosting
- Extra Trees
- HistGradient Boosting
- XGBoost

Notes:
- Replace 'path_to_structured_data.csv' with your dataset path.
- Target variable ('upr30') must be binary (0/1).
- Original EMR data cannot be shared due to privacy restrictions.
"""

```
# ============================================================
# 0. IMPORTS
# ============================================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.ensemble import (
    RandomForestClassifier,
    GradientBoostingClassifier,
    ExtraTreesClassifier,
    HistGradientBoostingClassifier
)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    roc_auc_score,
    precision_score,
    recall_score,
    f1_score,
    brier_score_loss,
    roc_curve,
    auc
)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Optional: XGBoost
try:
    from xgboost import XGBClassifier
    xgb_available = True
```

```
except ImportError:
    xgb_available = False


# ============================================================
# 1. LOAD DATA
# ============================================================
file_path = "path_to_structured_data.csv"
df = pd.read_csv(file_path)

# Ensure binary target
y = df["upr30"].astype(int)

# ============================================================
# 2. SELECT STRUCTURED FEATURES
# ============================================================
structured_cols = [
    "age", "sex", "HFRS", "NoofEDvisitsinprevious6mo",
    "Totalnumberofhospitaladmiss", "Charlsonindexcurrentadmission",
    "irsd", "crp", "smoking", "alcohol_abuse", "los"
]

structured_cols = [c for c in structured_cols if c in df.columns]

X = df[structured_cols].copy()
X = X.fillna(X.median())

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ============================================================
# 3. TRAIN / VALIDATION / TEST SPLIT (70 / 10 / 20)
# ============================================================
X_temp, X_test, y_temp, y_test = train_test_split(
    X_scaled, y, test_size=0.20, random_state=42, stratify=y
)

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.125, random_state=42, stratify=y_temp
)

y_train = y_train.values
y_val = y_val.values
y_test = y_test.values

# ============================================================
```

```
# 4. DEFINE ML MODELS
# ============================================================
ml_models = {
  "Logistic Regression": LogisticRegression(
    max_iter=1000, class_weight="balanced", random_state=42
  ),
  "Random Forest": RandomForestClassifier(
    n_estimators=500, random_state=42, class_weight="balanced"
  ),
  "Gradient Boosting": GradientBoostingClassifier(
    n_estimators=500, learning_rate=0.05, random_state=42
  ),
  "Extra Trees": ExtraTreesClassifier(
    n_estimators=500, random_state=42, class_weight="balanced"
  ),
  "HistGradient Boosting": HistGradientBoostingClassifier(
    max_iter=500, learning_rate=0.05, random_state=42
  )
}

if xgb_available:
  ml_models["XGBoost"] = XGBClassifier(
    n_estimators=500,
    learning_rate=0.05,
    use_label_encoder=False,
    eval_metric="logloss",
    random_state=42
  )

# ============================================================
# 5. TRAIN AND EVALUATE MODELS
# ============================================================
results = {}
model_probs = {}

for name, model in ml_models.items():
  print(f"\nTraining {name}...")
  model.fit(X_train, y_train)

  # Get probabilities safely
  if hasattr(model, "predict_proba"):
    test_probs = model.predict_proba(X_test)[:, 1]
  else:
    test_probs = model.decision_function(X_test)

  test_preds = (test_probs >= 0.5).astype(int)
  model_probs[name] = test_probs
```

```
    auc_score = roc_auc_score(y_test, test_probs)
    precision = precision_score(y_test, test_preds)
    recall = recall_score(y_test, test_preds)
    f1 = f1_score(y_test, test_preds)
    brier = brier_score_loss(y_test, test_probs)

    results[name] = {
        "AUC": auc_score,
        "Precision": precision,
        "Recall": recall,
        "F1": f1,
        "Brier": brier
    }

    print(
        f"{name} - "
        f"AUC: {auc_score:.4f}, "
        f"Precision: {precision:.4f}, "
        f"Recall: {recall:.4f}, "
        f"F1: {f1:.4f}, "
        f"Brier: {brier:.4f}"
    )

# ============================================================
# 6. PLOT ROC CURVES
# ============================================================
plt.figure(figsize=(8, 6))

for name, probs in model_probs.items():
    fpr, tpr, _ = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f"{name} (AUC = {roc_auc:.3f})")

plt.plot([0, 1], [0, 1], linestyle="--", lw=1, label="Random classifier")

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Classical ML Models (Structured Data)")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()

plt.savefig("ml_models_structured_roc.png", dpi=300)
plt.show()

print("\nROC figure saved as: ml_models_structured_roc.png")
```

"""
Unstructured Text Neural Network Model

This script implements a neural network for predicting 30-day unplanned
readmission using unstructured clinical text embeddings only
(e.g., Bio-ClinicalBERT embeddings).

Requirements:
- CSV file containing outcome variable 'upr30'
- NumPy file (.npy) containing text embeddings (n_samples x embedding_dim)

Note:
- Original EMR data cannot be shared due to privacy restrictions.
- Replace placeholder paths with your own dataset locations.
"""

```
# ============================================================
# 0. IMPORTS
# ============================================================
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Set reproducibility
torch.manual_seed(42)
np.random.seed(42)

# ============================================================
# 1. LOAD DATA
# ============================================================

# Structured file containing outcome only
data_path = "path_to_structured_data.csv"
df = pd.read_csv(data_path)

# Precomputed text embeddings (n_samples x embedding_dim)
embeddings_path = "path_to_text_embeddings.npy"
X_text = np.load(embeddings_path)

# Binary outcome (0/1)
```

```
y = df["upr30"].astype(int).values

print(f"Text embedding shape: {X_text.shape}")

# ============================================================
# 2. TRAIN / VALIDATION / TEST SPLIT (70 / 10 / 20)
# ============================================================

X_temp, X_test, y_temp, y_test = train_test_split(
   X_text, y, test_size=0.20, random_state=42, stratify=y
)

X_train, X_val, y_train, y_val = train_test_split(
   X_temp, y_temp, test_size=0.125, random_state=42, stratify=y_temp
)

print(f"Train: {len(X_train)}, Val: {len(X_val)}, Test: {len(X_test)}")

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)

X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val.reshape(-1, 1), dtype=torch.float32)

X_test = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32, shuffle=False)

# ============================================================
# 3. DEFINE UNSTRUCTURED TEXT MODEL
# ============================================================

class TextNN(nn.Module):
   def __init__(self, input_dim):
      super(TextNN, self).__init__()
      self.fc1 = nn.Linear(input_dim, 256)
      self.fc2 = nn.Linear(256, 64)
      self.fc3 = nn.Linear(64, 1)
      self.relu = nn.ReLU()
      self.sigmoid = nn.Sigmoid()

   def forward(self, x):
      x = self.relu(self.fc1(x))
      x = self.relu(self.fc2(x))
```

```
        return self.sigmoid(self.fc3(x))

model = TextNN(X_text.shape[1])

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# ============================================================
# 4. TRAINING WITH VALIDATION AUC + CHECKPOINTING
# ============================================================

epochs = 15
best_val_auc = 0

for epoch in range(epochs):
    model.train()
    train_loss = 0

    for xb, yb in train_loader:
        optimizer.zero_grad()
        preds = model(xb)
        loss = criterion(preds, yb)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    model.eval()
    val_preds = []

    with torch.no_grad():
        for xb, _ in val_loader:
            preds = model(xb)
            val_preds.extend(preds.numpy())

    val_auc = roc_auc_score(y_val, val_preds)

    print(
        f"Epoch {epoch+1}/{epochs} | "
        f"Train Loss: {train_loss:.4f} | "
        f"Val AUC: {val_auc:.4f}"
    )

    if val_auc > best_val_auc:
        best_val_auc = val_auc
        torch.save(model.state_dict(), "best_unstructured_model.pt")
        print("Best model updated.")
```

```
# ============================================================
# 5. FINAL TEST EVALUATION
# ============================================================

model.load_state_dict(torch.load("best_unstructured_model.pt"))
model.eval()

with torch.no_grad():
   test_probs = model(X_test).numpy()

test_auc = roc_auc_score(y_test, test_probs)

print("\n==============================")
print(f"FINAL TEST ROC-AUC: {test_auc:.4f}")
print("==============================")

# ============================================================
# 6. PLOT ROC CURVE
# ============================================================

fpr, tpr, _ = roc_curve(y_test, test_probs)

plt.figure(figsize=(7, 6))
plt.plot(fpr, tpr, lw=2, label=f"Text Model (AUC = {test_auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--", lw=1, label="Random classifier")

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Unstructured Text Model")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()

plt.savefig("unstructured_text_model_roc.png", dpi=300)
plt.show()

print("ROC figure saved as: unstructured_text_model_roc.png")

:Structured Data Neural Network

This script trains a feed-forward neural network using structured clinical variables
to predict 30-day outcome.

Data paths are placeholders. Replace with your local dataset.
"""

import pandas as pd
```

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# ==============================
# Reproducibility
# ==============================
torch.manual_seed(42)
np.random.seed(42)

# ==============================
# Load Dataset
# ==============================
file_path = "path_to_structured_dataset.csv"
df = pd.read_csv(file_path)

y = df['outcome_30day'].values

structured_features = [
   'age','sex','HFRS','ed_visits',
   'hospital_admissions','charlson_index',
   'irsd','crp','smoking','alcohol_use','los'
]

X = df[structured_features].values

# ==============================
# Standardization
# ==============================
scaler = StandardScaler()
X = scaler.fit_transform(X)

# ==============================
# Train / Validation / Test Split (70/10/20)
# ==============================
X_temp, X_test, y_temp, y_test = train_test_split(
   X, y, test_size=0.20, stratify=y, random_state=42
)

X_train, X_val, y_train, y_val = train_test_split(
   X_temp, y_temp, test_size=0.125, stratify=y_temp, random_state=42
)
```

```
# Convert to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train.reshape(-1,1), dtype=torch.float32)

X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val.reshape(-1,1), dtype=torch.float32)

X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test.reshape(-1,1), dtype=torch.float32)

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32)

# ==============================
# Model Definition
# ==============================
class StructuredNN(nn.Module):
  def __init__(self, input_dim):
    super().__init__()
    self.model = nn.Sequential(
      nn.Linear(input_dim, 256),
      nn.ReLU(),
      nn.Linear(256, 64),
      nn.ReLU(),
      nn.Linear(64, 1),
      nn.Sigmoid()
    )

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

model = StructuredNN(X.shape[1])
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# ==============================
# Training Loop
# ==============================
epochs = 10
best_val_auc = 0

for epoch in range(epochs):
  model.train()
  for xb, yb in train_loader:
    optimizer.zero_grad()
    preds = model(xb)
```

```
            loss = criterion(preds, yb)
            loss.backward()
            optimizer.step()

        model.eval()
        val_preds = []
        with torch.no_grad():
            for xb, _ in val_loader:
                val_preds.extend(model(xb).numpy())

        val_auc = roc_auc_score(y_val, val_preds)

        if val_auc > best_val_auc:
            best_val_auc = val_auc
            torch.save(model.state_dict(), "structured_model_best.pt")

        print(f"Epoch {epoch+1}/{epochs} - Val AUC: {val_auc:.4f}")

# ==============================
# Test Evaluation
# ==============================
model.load_state_dict(torch.load("structured_model_best.pt"))
model.eval()

with torch.no_grad():
    test_probs = model(X_test).numpy()

test_auc = roc_auc_score(y_test, test_probs)
print(f"Final Test AUC: {test_auc:.4f}")

fpr, tpr, _ = roc_curve(y_test, test_probs)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {test_auc:.3f}")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Structured Model")
plt.legend()
plt.savefig("structured_model_roc.png", dpi=300)
plt.close()
```

Multimodal Neural Network

```
Combines structured clinical variables with precomputed text embeddings.
Text embeddings were generated using a domain-specific transformer model.
"""

import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

torch.manual_seed(42)
np.random.seed(42)

# ==============================
# Load Structured Data
# ==============================
df = pd.read_csv("path_to_structured_dataset.csv")
text_embeddings = np.load("path_to_text_embeddings.npy")

y = df['outcome_30day'].values

structured_features = [
   'age','sex','HFRS','ed_visits',
   'hospital_admissions','charlson_index',
   'irsd','crp','smoking','alcohol_use','los'
]

X_struct = df[structured_features].values

scaler = StandardScaler()
X_struct = scaler.fit_transform(X_struct)

# Combine
X_combined = np.hstack([X_struct, text_embeddings])
input_dim = X_combined.shape[1]

# Split
X_train, X_test, y_train, y_test = train_test_split(
   X_combined, y, test_size=0.20, stratify=y, random_state=42
)

# Convert to tensor
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train.reshape(-1,1), dtype=torch.float32)
```

```
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test.reshape(-1,1), dtype=torch.float32)

# ==============================
# Model
# ==============================
class MultimodalNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 512),
            nn.ReLU(),
            nn.Linear(512, 128),
            nn.ReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

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

model = MultimodalNN(input_dim)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train
for epoch in range(15):
    model.train()
    optimizer.zero_grad()
    preds = model(X_train)
    loss = criterion(preds, y_train)
    loss.backward()
    optimizer.step()

# Test
model.eval()
with torch.no_grad():
    test_probs = model(X_test).numpy()

test_auc = roc_auc_score(y_test, test_probs)
print(f"Test AUC: {test_auc:.4f}")

"""
Supplementary Appendix C: Neural Network Using Clinical Variables

This script trains a feed-forward neural network to predict 30-day outcome
using structured clinical features.
```

```
Data paths are placeholders and must be replaced with local dataset paths.
Raw clinical data are not shared due to privacy regulations.
"""

# ============================================================
# Imports
# ============================================================
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# ============================================================
# Reproducibility
# ============================================================
torch.manual_seed(42)
np.random.seed(42)

# ============================================================
# 1. LOAD DATA
# ============================================================
file_path = "path_to_dataset.csv"   # <-- replace with your dataset
df = pd.read_csv(file_path)

y = df['outcome_30day'].values

feature_columns = [
   'age','sex','HFRS','ed_visits',
   'hospital_admissions','charlson_index',
   'irsd','crp','smoking','alcohol_use','los'
]

X = df[feature_columns].values

# ============================================================
# 2. STANDARDIZATION
# ============================================================
scaler = StandardScaler()
X = scaler.fit_transform(X)

# ============================================================
```

```
# 3. TRAIN / VALIDATION / TEST SPLIT (70/10/20)
# ============================================================
X_temp, X_test, y_temp, y_test = train_test_split(
   X, y, test_size=0.20, stratify=y, random_state=42
)

X_train, X_val, y_train, y_val = train_test_split(
   X_temp, y_temp, test_size=0.125, stratify=y_temp, random_state=42
)

print(f"Train: {len(X_train)}, Validation: {len(X_val)}, Test: {len(X_test)}")

# Convert to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train.reshape(-1,1), dtype=torch.float32)

X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val.reshape(-1,1), dtype=torch.float32)

X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test.reshape(-1,1), dtype=torch.float32)

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32)

# ============================================================
# 4. MODEL DEFINITION
# ============================================================
class ClinicalNN(nn.Module):
   def __init__(self, input_dim):
      super().__init__()
      self.network = nn.Sequential(
         nn.Linear(input_dim, 256),
         nn.ReLU(),
         nn.Linear(256, 64),
         nn.ReLU(),
         nn.Linear(64, 1),
         nn.Sigmoid()
      )

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

model = ClinicalNN(X.shape[1])
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
# ============================================================
# 5. TRAINING WITH VALIDATION AUC CHECKPOINTING
# ============================================================
epochs = 10
best_val_auc = 0

for epoch in range(epochs):

    # Training
    model.train()
    for xb, yb in train_loader:
        optimizer.zero_grad()
        preds = model(xb)
        loss = criterion(preds, yb)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    val_preds = []
    with torch.no_grad():
        for xb, _ in val_loader:
            val_preds.extend(model(xb).numpy())

    val_auc = roc_auc_score(y_val, val_preds)

    if val_auc > best_val_auc:
        best_val_auc = val_auc
        torch.save(model.state_dict(), "clinical_nn_best.pt")

    print(f"Epoch {epoch+1}/{epochs} - Validation AUC: {val_auc:.4f}")

# ============================================================
# 6. FINAL TEST EVALUATION
# ============================================================
model.load_state_dict(torch.load("clinical_nn_best.pt"))
model.eval()

with torch.no_grad():
    test_probs = model(X_test).numpy()

test_auc = roc_auc_score(y_test, test_probs)
print(f"\nFinal Test ROC-AUC: {test_auc:.4f}")

# ============================================================
# 7. ROC CURVE
# ============================================================
```

```
fpr, tpr, _ = roc_curve(y_test, test_probs)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {test_auc:.3f}")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Clinical Neural Network")
plt.legend()
plt.savefig("clinical_nn_roc.png", dpi=300)
plt.close()
```

Combined model using CNN

```
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
   roc_auc_score, accuracy_score, precision_score,
   recall_score, f1_score, brier_score_loss,
   confusion_matrix, roc_curve, auc
)
import matplotlib.pyplot as plt
import os

# ============================================================
# DEVICE
# ============================================================
device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
print("Using device:", device)

# ============================================================
# LOAD DATA
# ============================================================
file_path = "path_to_dataset.csv"   # <-- change to your dataset
df = pd.read_csv(file_path)

# ============================================================
# STRUCTURED DATA
# ============================================================
structured_cols = [
   'age', 'sex', 'Charlsonindexcurrentadmission', 'HFRS',
```

```
    'NoofEDvisitsinprevious6mo', 'Totalnumberofhospitaladmiss',
    'los', 'crp', 'irsd', 'smoking', 'alcohol_abuse'
]

X_structured = df[structured_cols].copy()
y = df['upr30']

# One-hot encode categorical variables
categorical_cols = ['sex', 'smoking', 'alcohol_abuse']
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

X_cat = ohe.fit_transform(X_structured[categorical_cols])
X_num = X_structured.drop(columns=categorical_cols).values

# Standardize numeric features
scaler = StandardScaler()
X_num = scaler.fit_transform(X_num)

# Combine numeric + categorical
X_structured = np.hstack([X_num, X_cat])

# ============================================================
# TRAIN / VAL / TEST SPLIT (70/10/20)
# ============================================================
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_structured, y, test_size=0.20, stratify=y, random_state=42
)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.125,
    stratify=y_train_val, random_state=42
)

# ============================================================
# UNSTRUCTURED TEXT (PLACEHOLDER EMBEDDINGS)
# ============================================================
np.random.seed(42)
X_train_text = np.random.randn(len(X_train), 768).astype(np.float32)
X_val_text   = np.random.randn(len(X_val), 768).astype(np.float32)
X_test_text  = np.random.randn(len(X_test), 768).astype(np.float32)

# ============================================================
# COMBINE STRUCTURED + TEXT
# ============================================================
X_train_combined = np.hstack([X_train, X_train_text])
X_val_combined   = np.hstack([X_val, X_val_text])
X_test_combined  = np.hstack([X_test, X_test_text])
```

```
# ============================================================
# PYTORCH DATASET
# ============================================================
class CombinedDataset(Dataset):
  def __init__(self, X, y):
    self.X = torch.tensor(X, dtype=torch.float32)
    self.y = torch.tensor(y.values, dtype=torch.float32)

  def __len__(self):
    return len(self.y)

  def __getitem__(self, idx):
    return self.X[idx], self.y[idx]

train_loader = DataLoader(CombinedDataset(X_train_combined, y_train),
                batch_size=32, shuffle=True)
val_loader = DataLoader(CombinedDataset(X_val_combined, y_val),
               batch_size=32)
test_loader = DataLoader(CombinedDataset(X_test_combined, y_test),
                batch_size=32)

# ============================================================
# CNN MODEL
# ============================================================
class CombinedCNN(nn.Module):
  def __init__(self, input_dim):
    super().__init__()
    self.conv1 = nn.Conv1d(1, 32, kernel_size=3, padding=1)
    self.relu = nn.ReLU()
    self.pool = nn.AdaptiveMaxPool1d(1)
    self.fc1 = nn.Linear(32, 64)
    self.fc2 = nn.Linear(64, 1)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    x = x.unsqueeze(1)
    x = self.conv1(x)
    x = self.relu(x)
    x = self.pool(x)
    x = x.squeeze(-1)
    x = self.relu(self.fc1(x))
    x = self.sigmoid(self.fc2(x))
    return x.squeeze()

input_dim = X_train_combined.shape[1]
model = CombinedCNN(input_dim).to(device)
```

```
# ============================================================
# LOSS & OPTIMIZER
# ============================================================
loss_fn = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# ============================================================
# TRAINING
# ============================================================
epochs = 10
best_val_auc = 0

for epoch in range(epochs):

    model.train()
    train_losses = []

    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        preds = model(X_batch)
        loss = loss_fn(preds, y_batch)
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())

    avg_train_loss = np.mean(train_losses)

    # Validation
    model.eval()
    val_preds, val_true = [], []

    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            X_batch = X_batch.to(device)
            preds = model(X_batch)

            val_preds.extend(preds.cpu().numpy())
            val_true.extend(y_batch.numpy())

    val_auc = roc_auc_score(val_true, val_preds)

    if val_auc > best_val_auc:
        best_val_auc = val_auc
```

```
        torch.save(model.state_dict(), "combined_cnn_best.pt")

    print(f"Epoch {epoch+1}/{epochs} - Train Loss: {avg_train_loss:.4f} - Val AUC:
{val_auc:.4f}")

print(f"\nBest Validation AUC: {best_val_auc:.4f}")

# ============================================================
# TEST EVALUATION
# ============================================================
model.load_state_dict(torch.load("combined_cnn_best.pt"))
model.eval()

test_preds, test_true = [], []

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch = X_batch.to(device)
        preds = model(X_batch)

        test_preds.extend(preds.cpu().numpy())
        test_true.extend(y_batch.numpy())

# Metrics
auc_score = roc_auc_score(test_true, test_preds)

print("\n--- CNN Combined Model Test Metrics ---")
print(f"AUC: {auc_score:.4f}")

# ============================================================
# ROC CURVE
# ============================================================
fpr, tpr, _ = roc_curve(test_true, test_preds)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {auc_score:.3f}")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Combined CNN")
plt.legend()
plt.show()
```

Combined model using LSTM

```
import torch
from torch import nn
```

```
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
   roc_auc_score, accuracy_score, precision_score,
   recall_score, f1_score, brier_score_loss,
   confusion_matrix, roc_curve, auc
)
import matplotlib.pyplot as plt
import os

# ============================================================
# DEVICE
# ============================================================
device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
print("Using device:", device)

# ============================================================
# LOAD DATA
# ============================================================
file_path = "path_to_dataset.csv"  # <-- change this
df = pd.read_csv(file_path)

# ============================================================
# STRUCTURED DATA
# ============================================================
structured_cols = [
   'age','sex','Charlsonindexcurrentadmission','HFRS',
   'NoofEDvisitsinprevious6mo','Totalnumberofhospitaladmiss',
   'los','crp','irsd','smoking','alcohol_abuse'
]

X_structured = df[structured_cols].copy()
y = df['upr30']

# One-hot encode categorical variables
categorical_cols = ['sex', 'smoking', 'alcohol_abuse']
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

X_cat = ohe.fit_transform(X_structured[categorical_cols])
X_num = X_structured.drop(columns=categorical_cols).values

# Standardize numeric features
scaler = StandardScaler()
X_num = scaler.fit_transform(X_num)
```

```
X_structured = np.hstack([X_num, X_cat])

# ============================================================
# TRAIN / VAL / TEST SPLIT (70/10/20)
# ============================================================
X_train_val, X_test, y_train_val, y_test = train_test_split(
   X_structured, y, test_size=0.20, stratify=y, random_state=42
)

X_train, X_val, y_train, y_val = train_test_split(
   X_train_val, y_train_val, test_size=0.125,
   stratify=y_train_val, random_state=42
)

# ============================================================
# SIMULATED TEXT EMBEDDINGS (Replace with real embeddings)
# ============================================================
np.random.seed(42)

X_train_text = np.random.randn(len(X_train), 768).astype(np.float32)
X_val_text   = np.random.randn(len(X_val), 768).astype(np.float32)
X_test_text  = np.random.randn(len(X_test), 768).astype(np.float32)

# ============================================================
# COMBINE STRUCTURED + TEXT
# ============================================================
X_train_combined = np.hstack([X_train, X_train_text])
X_val_combined   = np.hstack([X_val, X_val_text])
X_test_combined  = np.hstack([X_test, X_test_text])

# ============================================================
# PYTORCH DATASET
# ============================================================
class CombinedDataset(Dataset):
   def __init__(self, X, y):
      self.X = torch.tensor(X, dtype=torch.float32)
      self.y = torch.tensor(y.values, dtype=torch.float32)

   def __len__(self):
      return len(self.y)

   def __getitem__(self, idx):
      return self.X[idx], self.y[idx]

train_loader = DataLoader(CombinedDataset(X_train_combined, y_train),
                  batch_size=32, shuffle=True)
```

```
val_loader = DataLoader(CombinedDataset(X_val_combined, y_val),
                        batch_size=32)
test_loader = DataLoader(CombinedDataset(X_test_combined, y_test),
                         batch_size=32)

# ============================================================
# LSTM MODEL
# ============================================================
class CombinedLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim=128):
        super().__init__()
        self.lstm = nn.LSTM(input_size=input_dim,
                            hidden_size=hidden_dim,
                            batch_first=True)
        self.fc1 = nn.Linear(hidden_dim, 64)
        self.fc2 = nn.Linear(64, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = x.unsqueeze(1)  # (batch, seq_len=1, features)
        _, (hn, _) = self.lstm(x)
        x = self.relu(self.fc1(hn[-1]))
        x = self.sigmoid(self.fc2(x))
        return x.squeeze()

input_dim = X_train_combined.shape[1]
model = CombinedLSTM(input_dim).to(device)

# ============================================================
# LOSS & OPTIMIZER
# ============================================================
loss_fn = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# ============================================================
# TRAINING LOOP
# ============================================================
epochs = 10
best_val_auc = 0

for epoch in range(epochs):

    model.train()
    train_losses = []

    for X_batch, y_batch in train_loader:
```

```
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        preds = model(X_batch)
        loss = loss_fn(preds, y_batch)
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())

    avg_train_loss = np.mean(train_losses)

    # Validation
    model.eval()
    val_preds, val_true = [], []

    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            X_batch = X_batch.to(device)
            preds = model(X_batch)

            val_preds.extend(preds.cpu().numpy())
            val_true.extend(y_batch.numpy())

    val_auc = roc_auc_score(val_true, val_preds)

    if val_auc > best_val_auc:
        best_val_auc = val_auc
        torch.save(model.state_dict(), "combined_lstm_best.pt")

    print(f"Epoch {epoch+1}/{epochs} - Train Loss: {avg_train_loss:.4f} - Val AUC:
{val_auc:.4f}")

print(f"\nBest Validation AUC: {best_val_auc:.4f}")

# ============================================================
# TEST EVALUATION
# ============================================================
model.load_state_dict(torch.load("combined_lstm_best.pt"))
model.eval()

test_preds, test_true = [], []

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch = X_batch.to(device)
        preds = model(X_batch)
```

```
        test_preds.extend(preds.cpu().numpy())
        test_true.extend(y_batch.numpy())

# Metrics
auc_score = roc_auc_score(test_true, test_preds)

print("\n--- LSTM Combined Model Test Metrics ---")
print(f"AUC: {auc_score:.4f}")

# ============================================================
# ROC CURVE
# ============================================================
fpr, tpr, _ = roc_curve(test_true, test_preds)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {auc_score:.3f}")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Combined LSTM")
plt.legend()
plt.show()
```