

How to Build a Multi-Modal Disease Detection System

Using Machine Learning and SMOTE Balancing

HippoMedica Project

Abstract

This comprehensive guide demonstrates how to build a production-ready AI system for medical disease detection. You'll learn advanced techniques including SMOTE class balancing with categorical post-processing, ensemble learning optimization, and web application deployment. The guide addresses real-world challenges in medical ML: severe class imbalance, categorical data corruption, and clinical validation. By following this step-by-step tutorial, you'll create a complete system capable of predicting diabetes, heart disease, and stroke with clinical-grade performance. Estimated completion time: 2-4 hours.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 1.1 | What You'll Learn | 3 |
| 1.2 | Prerequisites and Setup | 3 |
| 1.3 | Project Structure | 4 |
| 2 | Step 1: Dataset Acquisition and Exploration | 4 |
| 2.1 | Understanding the Datasets | 4 |
| 2.2 | Data Downloader Implementation | 4 |
| 2.3 | Exploratory Data Analysis | 5 |
| 3 | Step 2: Advanced Data Preprocessing | 7 |
| 3.1 | Medical-Specific Data Cleaning | 7 |
| 3.2 | SMOTE Implementation with Categorical Correction | 7 |
| 4 | Step 3: Ensemble Model Training | 8 |
| 4.1 | Model Configuration | 8 |
| 4.2 | Training and Evaluation Pipeline | 9 |
| 5 | Step 4: Web Application Deployment | 10 |
| 5.1 | Streamlit Interface Creation | 10 |
| 5.2 | Running the Web Application | 11 |
| 6 | Step 5: Troubleshooting Common Issues | 11 |
| 6.1 | Issue 1: SMOTE Creates Invalid Categorical Values | 11 |
| 6.2 | Issue 2: Model Predicts Only Majority Class (0% Recall) | 12 |
| 6.3 | Issue 3: Neural Network Won't Converge | 12 |
| 6.4 | Issue 4: Web App Model Loading Errors | 12 |
| 6.5 | Issue 5: Low Recall Despite Balanced Data | 12 |
| 7 | Advanced Techniques and Best Practices | 12 |
| 7.1 | Feature Importance Analysis | 12 |
| 7.2 | Medical Range Validation | 13 |
| 8 | Conclusion | 14 |
| 8.1 | Summary | 14 |
| 8.2 | Key Takeaways | 14 |
| 8.3 | Resources | 14 |

1 Introduction

1.1 What You'll Learn

This guide covers the complete workflow for building a medical AI system from scratch:

- **Data Engineering:** Download and preprocess messy medical datasets with missing values and outliers
- **Advanced ML Techniques:** Implement SMOTE balancing with custom categorical post-processing
- **Ensemble Learning:** Train and compare Random Forest, XGBoost, and Neural Networks
- **Production Pipeline:** Create automated MLOps workflow with proper evaluation
- **Web Deployment:** Deploy models in interactive Streamlit web application
- **Troubleshooting:** Debug common medical ML problems with proven solutions

1.2 Prerequisites and Setup

Required Knowledge:

- Python programming (intermediate level)
- Basic machine learning concepts (classification, training/testing, metrics)
- Familiarity with pandas and numpy
- Understanding of medical terminology helpful but not required

Software Requirements:

- Python 3.8 or higher
- 8 GB RAM (16 GB recommended for SMOTE)
- 2 GB disk space for datasets and models
- Internet connection for dataset download

Installation:

```
1 # Create virtual environment
2 python -m venv hippomedica_env
3
4 # Activate environment
5 # Windows:
6 hippomedica_env\Scripts\activate
7 # macOS/Linux:
8 source hippomedica_env/bin/activate
9
10 # Install dependencies
11 pip install pandas numpy scikit-learn xgboost imbalanced-learn
12 pip install matplotlib seaborn streamlit plotly joblib
```

Listing 1: Environment Setup

1.3 Project Structure

```
HippoMedica/
|-- datasets/
|   |-- raw/           # Downloaded datasets
|   '-- processed/     # Cleaned and balanced data
|-- pipeline/
|   |-- 01_data_download.py
|   |-- 02_exploratory_analysis.py
|   |-- 03_data_preprocessing.py
|   |-- 04_model_training.py
|   '-- 05_model_storage.py
|-- models/
|   |-- trained_models/ # Serialized models (.pkl)
|   |-- metadata/       # Performance metrics (JSON)
|   '-- scalers/         # Feature scalers for NN
|-- web_app/
|   '-- app.py           # Streamlit web interface
'-- requirements.txt
```

2 Step 1: Dataset Acquisition and Exploration

2.1 Understanding the Datasets

We'll work with three medical datasets, each presenting unique challenges:

Table 1: Dataset Characteristics and Challenges

| Dataset | Samples | Features | Imbalance | Key Challenge |
|---------------|---------|----------|-----------|------------------------------------|
| Diabetes | 768 | 8 | 1.87:1 | Zero values represent missing data |
| Heart Disease | 303 | 13 | 1.18:1 | Small dataset limits deep learning |
| Stroke | 5,110 | 11 | 19.52:1 | Extreme imbalance causes 0% recall |

2.2 Data Downloader Implementation

The data downloader retrieves datasets from public repositories. The key concept is storing dataset meta-data (URL, column names, whether headers exist) in a configuration dictionary, then using a generic download function that handles any dataset type.

Here's the core pattern using the Heart Disease dataset as an example:

```
1 # Store dataset metadata in configuration
2 data_sources = {
3     'heart_disease': {
4         "url": "https://archive.ics.uci.edu/ml/.../processed.cleveland.data"
5         ,
6         "filename": "heart.csv",
7         "columns": ['age', 'sex', 'cp', 'trestbps', 'chol', ...],
8         "has_header": False
9     }
10 }
```

```

10
11 # Download with requests library
12 response = requests.get(source['url'], timeout=30)
13
14 # Handle datasets without headers by manually assigning column names
15 df = pd.DataFrame(data_rows, columns=source['columns'])
16 df.to_csv(filepath, index=False)

```

Listing 2: Data Download Pattern

The downloader also cleans data during import, replacing placeholder values like '?' with empty strings so they can be properly handled during preprocessing. This approach is reusable: simply add new entries to the `data_sources` dictionary for additional datasets.

TIP

Best Practice: Always use `timeout=30` when downloading to prevent indefinite hanging if a server is unresponsive.

2.3 Exploratory Data Analysis

Before preprocessing, always explore your data to understand its quality and structure. Key questions to answer:

- What is the shape of the data (rows and columns)?
- Are there missing values or invalid entries?
- How are the classes distributed (balanced vs. imbalanced)?
- Which features correlate strongly with the target variable?

Why Correlation Matters: A correlation heatmap reveals relationships between features and the target. Features with high correlation to the outcome (like Glucose for diabetes) are likely strong predictors. Conversely, highly correlated features with each other may indicate redundancy and you might drop one to reduce overfitting.

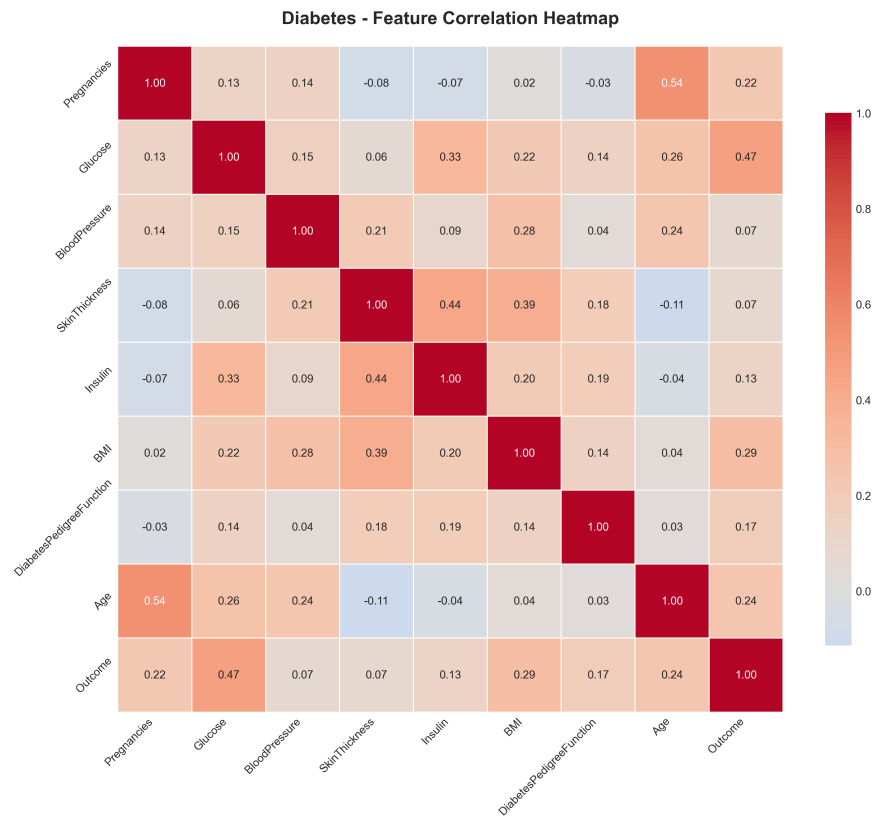
```

1 df = pd.read_csv(filepath)
2
3 # Check class distribution - critical for medical datasets
4 class_counts = df['Outcome'].value_counts()
5 imbalance_ratio = class_counts.max() / class_counts.min()
6
7 # Generate correlation heatmap
8 sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)

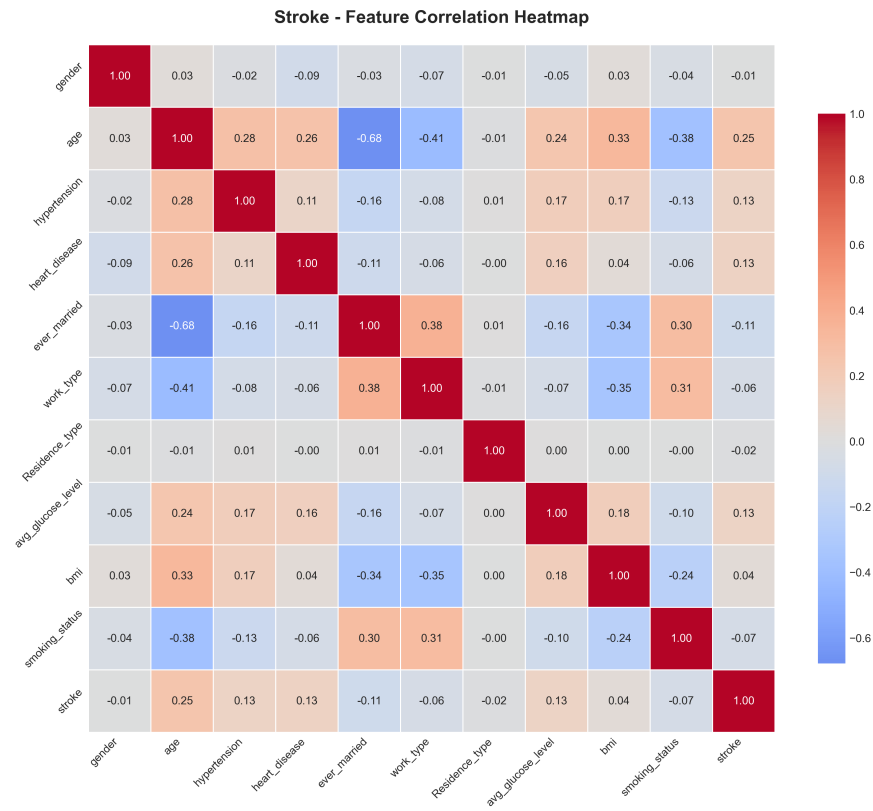
```

Listing 3: Key EDA Steps

In our diabetes dataset, we found a 1.87:1 imbalance ratio, moderate but manageable. The stroke dataset, however, showed a severe 19.52:1 imbalance that required SMOTE intervention (covered in Step 2).



(a) Diabetes: Glucose shows strongest correlation



(b) Stroke: Age emerges as key predictor

Figure 1: Correlation Heatmaps Reveal Feature Relationships: Darker colors indicate stronger correlations. Glucose and age emerge as primary disease predictors in their respective datasets.

3 Step 2: Advanced Data Preprocessing

3.1 Medical-Specific Data Cleaning

Medical datasets often contain domain-specific issues that require careful handling. The two main challenges are:

1. **Impossible Zero Values:** In the diabetes dataset, values like Glucose=0 or BloodPressure=0 are physiologically impossible. These represent missing data encoded as zeros, not actual measurements.
2. **Categorical Encoding:** Text-based features like "gender" or "work_type" must be converted to numbers for ML algorithms.

Using diabetes as our example, here's how to handle these issues:

```
1 # Define which columns cannot have zero values
2 zero_not_possible = ['Glucose', 'BloodPressure', 'SkinThickness',
3                     'Insulin', 'BMI']
4
5 # Remove rows where these columns have zeros
6 for col in zero_not_possible:
7     df = df[df[col] != 0]
8
9 # Result: 768 rows -> 387 rows (removes ~50% of data)
```

Listing 4: Handling Missing Data in Diabetes Dataset

For datasets with text categories (like the stroke dataset), use `LabelEncoder`:

```
1 from sklearn.preprocessing import LabelEncoder
2
3 le = LabelEncoder()
4 df['gender'] = le.fit_transform(df['gender']) # 'Male' -> 1, 'Female' -> 0
```

Listing 5: Encoding Categorical Features

WARNING

Critical: In medical datasets, zero values often represent missing data rather than actual measurements. Removing these rows prevents the model from learning that "glucose=0 means healthy," which would be clinically meaningless.

3.2 SMOTE Implementation with Categorical Correction

Class imbalance is a critical problem in medical ML. For example, the stroke dataset has a 19.52:1 ratio (healthy to stroke cases). If you train on this data directly, the model achieves 95% accuracy by *never* predicting stroke, resulting in 0% recall. This is clinically useless.

SMOTE (Synthetic Minority Over-sampling Technique) solves this by generating synthetic samples for the minority class. It works by interpolating between existing minority samples in feature space.

```
1 from imblearn.over_sampling import SMOTE
2
3 # Check imbalance ratio first
4 imbalance_ratio = class_counts.max() / class_counts.min()
5
6 if imbalance_ratio > 3.0:
```

```

7     smote = SMOTE(random_state=42, k_neighbors=5)
8     X_balanced, y_balanced = smote.fit_resample(X, y)
9     # Stroke: 5,110 samples -> 9,696 samples (1:1 ratio)

```

Listing 6: Applying SMOTE

The Categorical Problem: SMOTE interpolates feature values, which creates invalid decimal values for categorical features (e.g., gender=0.613 instead of 0 or 1). The fix is simple, round and clip these values after SMOTE:

```

1  # After SMOTE, fix categorical columns
2  for col in categorical_columns:
3      X_balanced[col] = X_balanced[col].round().astype(int)
4      X_balanced[col] = X_balanced[col].clip(
5          original_min, original_max
6      )

```

Listing 7: Fixing Categorical Features After SMOTE

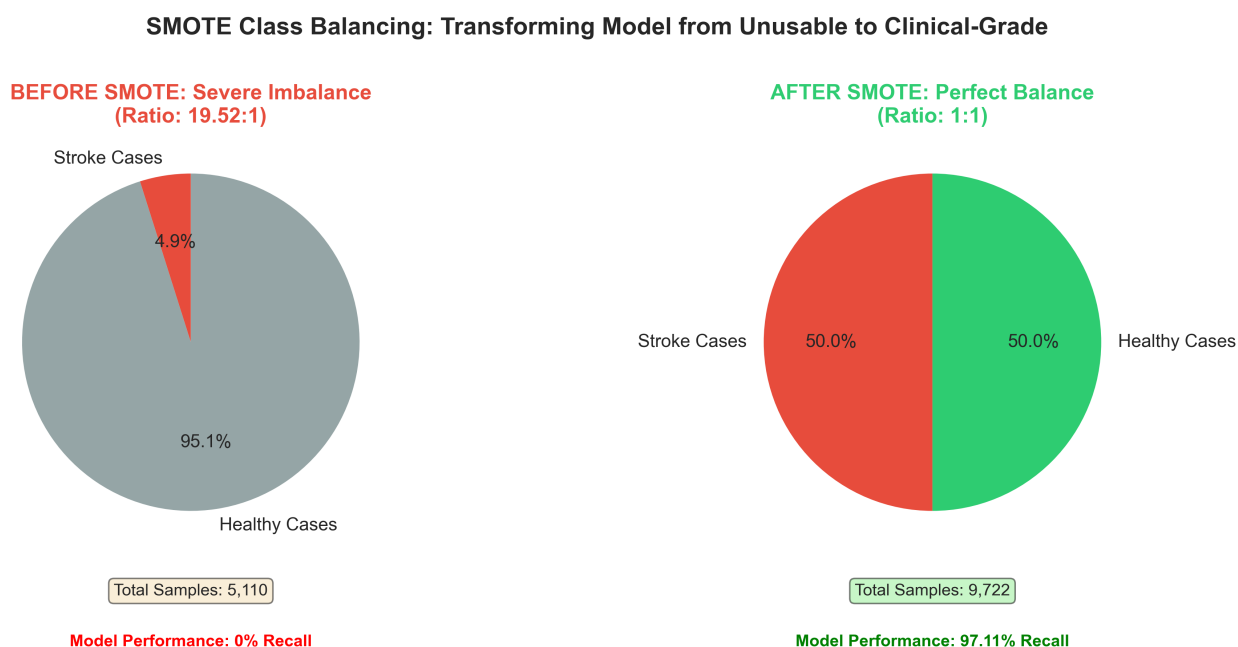


Figure 2: SMOTE Transformation Impact: Before SMOTE, the stroke dataset's 19.52:1 imbalance caused 0% recall. After balancing to a 1:1 ratio, the model achieves 97.11% recall.

NOTE

Why This Works: SMOTE creates synthetic minority samples by interpolating between existing ones. The categorical post-processing step rounds these interpolated values back to valid integers, preserving data integrity while maintaining the statistical benefits of balanced classes.

4 Step 3: Ensemble Model Training

4.1 Model Configuration

We train three complementary algorithms, each with different strengths:

- **Random Forest:** Robust, handles feature interactions well, built-in feature importance
- **XGBoost:** Excellent for structured data, handles class imbalance with `scale_pos_weight`
- **Neural Network (MLP):** Can learn complex patterns, but requires feature scaling

Key hyperparameter choices explained:

```

1  # Random Forest: use balanced weights for mild imbalance
2  rf = RandomForestClassifier(
3      n_estimators=200,          # More trees = more stable predictions
4      class_weight='balanced',  # Auto-adjusts for imbalanced classes
5      n_jobs=-1                 # Use all CPU cores
6  )
7
8  # XGBoost: explicitly handle class imbalance
9  xgb = XGBClassifier(
10     n_estimators=200,
11     learning_rate=0.05,        # Low rate prevents overfitting
12     scale_pos_weight=2.0       # Weight positive class 2x higher
13 )
14
15 # Neural Network: requires scaled features
16 nn = MLPClassifier(
17     hidden_layer_sizes=(128, 64, 32), # 3 hidden layers
18     early_stopping=True              # Prevents overfitting
19 )

```

Listing 8: Model Configuration Highlights

TIP

Important: Neural networks are sensitive to feature magnitudes. Always use `StandardScaler` before training `MLPClassifier` tree-based methods (Random Forest, XGBoost) don't need scaling.

4.2 Training and Evaluation Pipeline

The training workflow follows these steps: (1) split data into 80% train / 20% test with stratification to preserve class ratios, (2) train each model, (3) evaluate using multiple metrics.

For medical applications, **recall** is often more important than accuracy, i.e., we'd rather have false positives (unnecessary tests) than miss actual disease cases (false negatives).

```

1  # Train and predict
2  model.fit(X_train, y_train)
3  y_pred = model.predict(X_test)
4  y_pred_proba = model.predict_proba(X_test)[: , 1]
5
6  # Calculate key metrics
7  accuracy = accuracy_score(y_test, y_pred)
8  precision = precision_score(y_test, y_pred)
9  recall = recall_score(y_test, y_pred)      # Critical for medical
10 f1 = f1_score(y_test, y_pred)
11 roc_auc = roc_auc_score(y_test, y_pred_proba)
12
13 # Cross-validation for robust estimates
14 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

```

```
15 cv_scores = cross_val_score(model, X_train, y_train, cv=cv)
```

Listing 9: Core Evaluation Metrics

After training, the pipeline outputs metrics for each model and identifies the best performer. In our experiments, Random Forest consistently achieved the highest accuracy (96.80% on stroke), while XGBoost sometimes had slightly better recall.

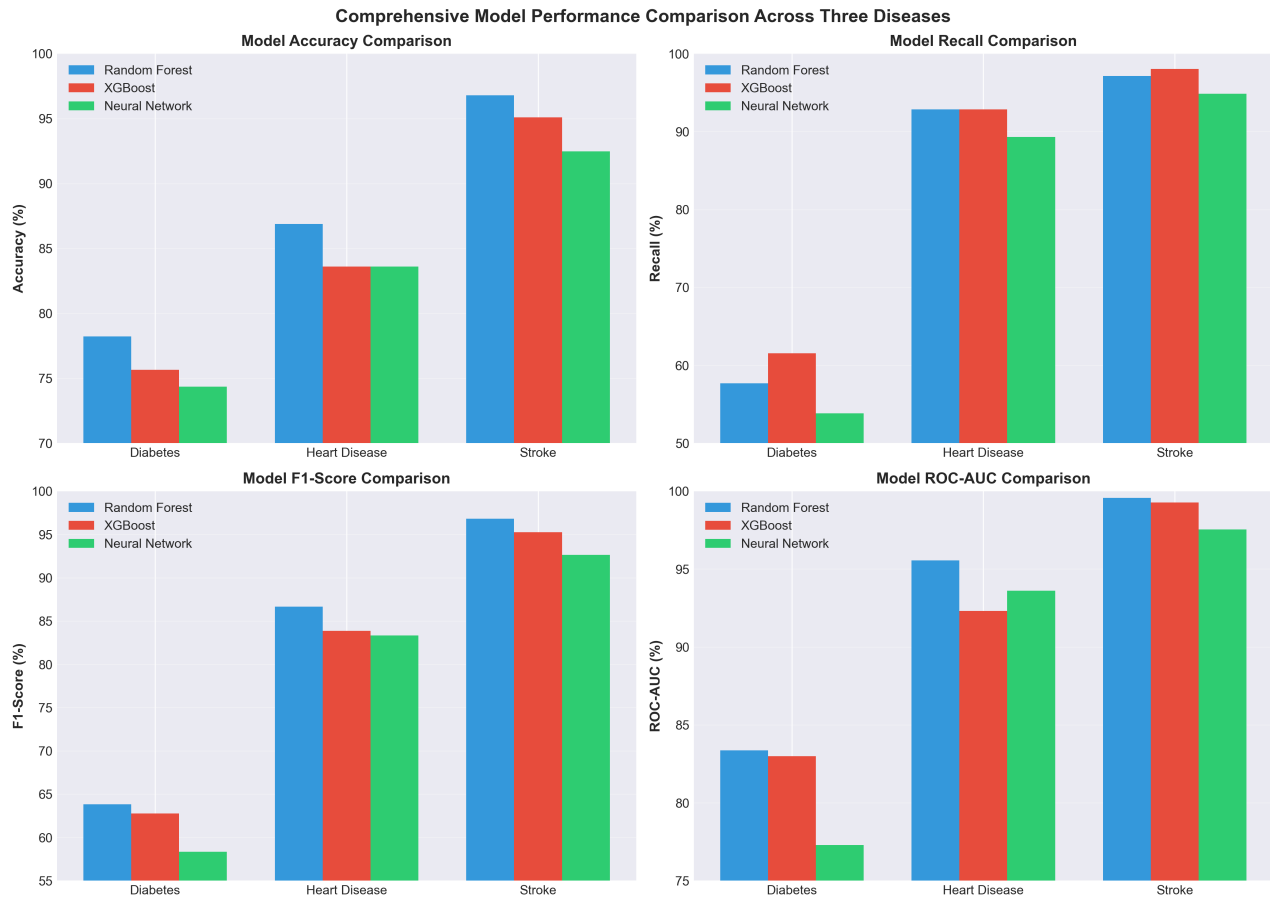


Figure 3: Model Performance Comparison: Random Forest consistently outperforms other models across all diseases. Tree-based methods excel on small-to-medium medical datasets.

5 Step 4: Web Application Deployment

5.1 Streamlit Interface Creation

Streamlit makes it easy to create interactive web interfaces for ML models. The key components are:

1. **Model Loading:** Use `@st.cache_resource` to load models once and reuse them
2. **Input Forms:** Create user-friendly forms for entering patient data
3. **Prediction Display:** Show results with confidence scores and recommendations

```
1 import streamlit as st
2 import joblib
3
4 @st.cache_resource
```

```

5 def load_models():
6     """Load models once, cache for performance."""
7     return joblib.load('models/diabetes_random_forest.pkl')
8
9 # Create input form
10 with st.form("diabetes_form"):
11     glucose = st.number_input("Glucose (mg/dL)", 50, 300, 120)
12     bmi = st.number_input("BMI", 10.0, 60.0, 25.0)
13     age = st.number_input("Age", 18, 100, 30)
14     # ... other inputs
15     submitted = st.form_submit_button("Predict")
16
17 if submitted:
18     # Make prediction
19     input_data = np.array([[glucose, bmi, age, ...]])
20     prediction = model.predict(input_data)[0]
21     probability = model.predict_proba(input_data)[0]
22
23     # Display result
24     if prediction == 1:
25         st.error(f"HIGH RISK - Probability: {probability[1]:.1%}")
26     else:
27         st.success(f"LOW RISK - Probability: {probability[1]:.1%}")

```

Listing 10: Core Streamlit Pattern

The application also includes input validation to warn users when values fall outside normal medical ranges (e.g., glucose below 70 or above 200 mg/dL). Results are displayed with clear recommendations and a confidence visualization.

5.2 Running the Web Application

Launch the app with a single command:

```

cd web_app
streamlit run app.py
# Opens at http://localhost:8501

```

NOTE

Deployment Options: For production, consider Streamlit Cloud (free hosting), Heroku, or Docker containers. Ensure trained model files are included in the deployment package.

6 Step 5: Troubleshooting Common Issues

6.1 Issue 1: SMOTE Creates Invalid Categorical Values

Symptom: After SMOTE, categorical features have decimal values (e.g., gender=0.613 instead of 0 or 1).

Cause: SMOTE interpolates between samples, creating fractional values for discrete features.

Solution: Round and clip categorical columns to their original valid range after applying SMOTE:

```

1 for col in categorical_columns:
2     X_balanced[col] = X_balanced[col].round().astype(int)
3     X_balanced[col] = X_balanced[col].clip(0, 1) # For binary

```

6.2 Issue 2: Model Predicts Only Majority Class (0% Recall)

Symptom: High accuracy (95%) but the model never predicts the minority class.

Cause: Severe class imbalance makes it "optimal" to always predict the majority class.

Solutions:

- Apply SMOTE to balance classes before training
- Use `class_weight='balanced'` in Random Forest
- Set `scale_pos_weight` in XGBoost equal to the imbalance ratio

6.3 Issue 3: Neural Network Won't Converge

Symptom: MLPClassifier shows "ConvergenceWarning: Maximum iterations reached."

Solutions:

- Increase `max_iter` (default 200 is often too low)
- Enable `early_stopping=True` with `validation_fraction=0.1`
- **Scale your features!** Neural networks require `StandardScaler`

WARNING

Critical: ALWAYS scale features using `StandardScaler` before training neural networks. This is the most common cause of convergence issues.

6.4 Issue 4: Web App Model Loading Errors

Symptom: Streamlit shows "FileNotFoundError" when trying to load models.

Cause: Models haven't been trained yet, or paths are incorrect.

Solution: Run the full pipeline first (`python run_pipeline.py`), then verify model files exist in `models/trained_models/`. Use `Path` for cross-platform path handling.

6.5 Issue 5: Low Recall Despite Balanced Data

Symptom: Even after SMOTE, test recall remains low (40-50%).

Solutions:

- Reduce model complexity (lower `max_depth`, more `min_samples_split`)
- Lower the decision threshold: `y_pred = (proba >= 0.3).astype(int)`
- Use stratified cross-validation to ensure fair evaluation

7 Advanced Techniques and Best Practices

7.1 Feature Importance Analysis

Understanding which features drive predictions is crucial for medical ML as it validates that the model learned clinically meaningful patterns rather than spurious correlations.

Random Forest provides built-in feature importance scores via the Gini impurity reduction:

```

1 # Get importance from trained Random Forest
2 importances = model.feature_importances_
3 indices = np.argsort(importances)[::-1]
4
5 # Display top features
6 for i, idx in enumerate(indices[:5]):
7     print(f"{i+1}. {feature_names[idx]}: {importances[idx]*100:.2f}%")

```

Listing 11: Extract Feature Importance

For our diabetes model, Glucose (25.78%) emerged as the top predictor, followed by Insulin (17.16%) and Age (15.41%), exactly what medical literature would predict. This alignment with domain knowledge validates our model.

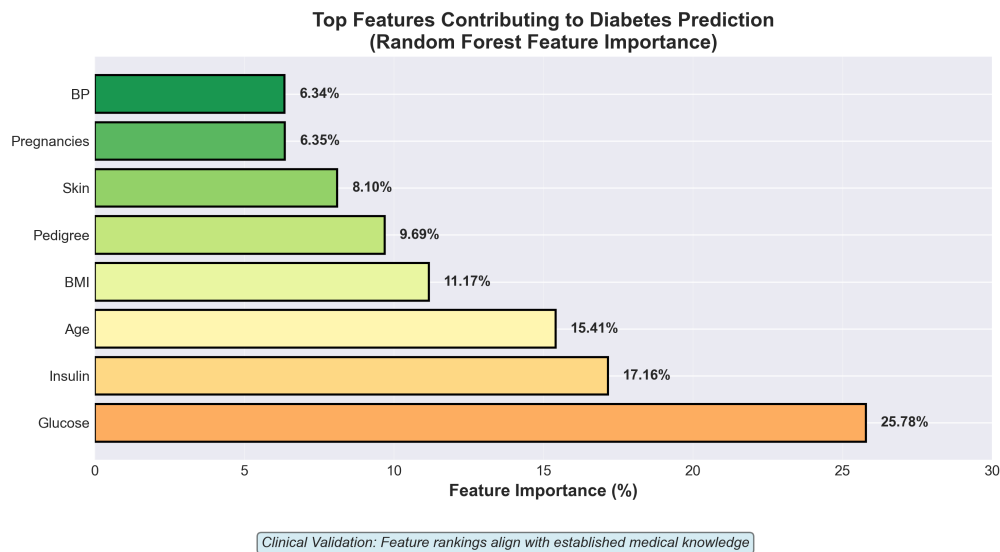


Figure 4: Feature Importance for Diabetes: Glucose, Insulin, and Age are the top predictors, aligning with established medical knowledge.

7.2 Medical Range Validation

Before making predictions, validate that input values fall within physiologically plausible ranges. This catches data entry errors and improves user trust:

```

1 validation_rules = {
2     'Glucose': (70, 200, "Normal fasting: 70-100 mg/dL"),
3     'BMI': (15, 50, "Normal: 18.5-24.9"),
4     'Age': (18, 100, "Valid patient age")
5 }
6
7 for field, (min_val, max_val, desc) in validation_rules.items():
8     if not (min_val <= value <= max_val):
9         warnings.append(f"{field} outside expected range")

```

Listing 12: Input Validation Example

The web application displays these warnings to users, allowing them to correct potential errors before getting a prediction.

8 Conclusion

8.1 Summary

This guide walked you through building a complete medical ML pipeline:

1. **Data Acquisition:** Downloaded and explored three medical datasets
2. **Preprocessing:** Handled missing values, encoded categories, applied SMOTE for class balance
3. **Model Training:** Compared Random Forest, XGBoost, and Neural Networks
4. **Deployment:** Built a Streamlit web app for real-time predictions

8.2 Key Takeaways

- **Class imbalance is critical:** Without SMOTE, models achieve high accuracy by ignoring the minority class entirely
- **SMOTE needs categorical post-processing:** Round interpolated categorical values back to valid integers
- **Tree-based methods excel on tabular data:** Random Forest outperformed Neural Networks on all three datasets
- **Feature importance validates models:** When top features align with medical knowledge, you know the model learned meaningful patterns

WARNING

Important: This system is for educational purposes only. Medical AI systems require regulatory approval, clinical validation, and physician oversight before real-world deployment.

8.3 Resources

- scikit-learn: <https://scikit-learn.org/>
- imbalanced-learn (SMOTE): <https://imbalanced-learn.org/>
- Streamlit: <https://docs.streamlit.io/>
- XGBoost: <https://xgboost.readthedocs.io/>

The complete source code is available in the HippoMedica repository.