Project Report : Medical Diagnosis Expert System - An expert system for medical diagnosis that assists healthcare professionals. Course: CSE440, Section:01, Group:03

Md Tamim Ahamed:201426042, Abdul Zabbar Dewan:2013222042, Mir Bayazid Protik:1921085042, Sayeb Rayat:2012733042. Email address: tamim.ahamed@nothsouth.edu

Abstract

The Medical Diagnosis Expert System (MDES) is an intelligent, hybrid diagnostic tool meticulously designed to assist healthcare professionals in delivering precise and timely medical diagnoses. By synergizing the power of machine learning algorithms with rule-based inferential logic, MDES analyzes patient-reported symptoms and historical medical data to generate a list of probable conditions, each accompanied by a confidence score that quantifies the system's certainty. This dual-approach not only enhances diagnostic reliability but also provides transparent reasoning behind each suggestion.

At the heart of MDES lie three robust machine learning models — Decision Tree, k-Nearest Neighbors (kNN), and Support Vector Machine (SVM) — each selected for their proven efficacy in classification tasks within clinical datasets. Prior to model training, patient data undergoes Min-Max feature scaling to ensure uniformity and improve model convergence.

To foster interpretability and facilitate exploratory data analysis, the system integrates a diverse suite of data visualization techniques. These include scatter plots to identify trends, Kernel Density Estimation (KDE) plots to assess probability distributions, histograms for frequency analysis, and swarm, violin, and strip plots to reveal intricate patterns, anomalies, and the spread of symptom variables across diagnostic categories. These visualizations serve as an intuitive bridge between raw clinical data and actionable medical insight.

Purpose-built for seamless adoption in real-world clinical environments, MDES aspires to augment diagnostic decision-making, reduce the incidence of human error, and ultimately contribute to improved patient outcomes. Looking ahead, the system's capabilities are envisioned to evolve through the incorporation of real-time patient monitoring data, natural language processing for unstructured inputs, and an ever-expanding medical knowledge base to adapt to emerging health trends and complex diagnostic challenges.

Methodology

1. Problem Definition

The increasing complexity of medical diagnostics, particularly in scenarios involving multifactorial conditions such as diabetes and cardiovascular complications, places immense pressure on healthcare professionals. Diagnoses often require careful evaluation of a wide range of physiological and historical variables including blood pressure, heart rate, mental health status, and risk indicators. Relying solely on manual assessment can lead to delays, oversight, or inconsistencies—especially when patient volumes are high or clinical histories are incomplete.

This project addresses the urgent need for an intelligent, automated diagnostic system that can analyze multiple patient attributes simultaneously and suggest reliable, data-driven predictions. By leveraging machine learning and expert knowledge, the system aims to support clinicians in identifying underlying health risks and recommending possible diagnoses with high confidence, particularly in patients exhibiting signs of chronic or high-risk conditions.

Objectives

- **Develop a Medical Diagnosis Expert System (MDES)** that utilizes 12 key clinical variables — *Age, Systolic and Diastolic Blood Pressure, Blood Sugar (BS), Body Temperature, BMI, Heart Rate, Previous Complications, Preexisting Diabetes, Gestational Diabetes, Mental Health*, and *Risk Level* — to assist healthcare professionals in making accurate diagnostic predictions.
- **Implement and evaluate machine learning models** including **Decision Tree**, **k-Nearest Neighbors (kNN)**, and **Support Vector Machine (SVM)** for classification tasks based on patient data.
- **Apply Min-Max feature scaling** to normalize input data and improve the consistency and convergence behavior of machine learning algorithms.
- **Utilize advanced data visualization techniques** such as **scatter plots**, **KDE plots**, **histograms**, **swarm plots**, **violin plots**, and **strip plots** to explore data patterns and enhance interpretability.
- **Generate interpretable predictions with confidence scores** to support medical decision-making and provide transparency in diagnostic reasoning.
- **Design a user-friendly interface** that enables seamless clinical integration and supports real-time diagnostic assistance.

Challenges

- **Complex feature interactions**: Variables such as *mental health*, *previous complications*, and *diabetes history* may influence each other in non-linear ways, requiring careful modeling and analysis.
- **Imbalanced and noisy data**: Medical datasets often contain class imbalance and missing or inconsistent entries, which can affect model accuracy and bias predictions.
- **Risk stratification complexity**: Differentiating between *low, medium, and high risk* patients requires subtle, clinically informed thresholds that are challenging to model.
- **Model interpretability**: Some machine learning algorithms, especially *SVM*, act as black boxes, making it difficult to explain predictions to medical professionals without additional tools.
- **Generalization across populations**: The system must perform reliably across diverse demographic and physiological profiles, not just the training set population.
- **Ethical and legal compliance**: Ensuring **data privacy**, **security**, and adherence to medical regulations (e.g., HIPAA, GDPR) is critical when handling sensitive health information.

I. Data Collection and Processing

This study utilizes a structured medical dataset comprising 12 essential clinical variables related to patient health and risk assessment. The dataset, stored in CSV format and titled `Dataset - Updated.csv`, was imported and processed using Python with the aid of scientific libraries such as `pandas`, `numpy`, `matplotlib`, and `seaborn`. These libraries facilitated efficient data handling, preprocessing, and visualization.

The dataset includes the following variables:

- Age
- Systolic Blood Pressure (Systolic BP)
- Diastolic Blood Pressure (Diastolic)
- Blood Sugar (BS)
- Body Temperature
- Body Mass Index (BMI)
- Previous Complications
- Preexisting Diabetes
- Gestational Diabetes
- Mental Health
- Heart Rate
- Risk Level (Target variable)

The data loading and initial exploration steps were executed using the following Python code:

Listing 1: Data Loading and Inspection

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('Dataset␣-␣Updated.csv')
# Display DataStructure & SampleEntries
df.info()
df.head()
```

This initial processing phase ensures that the dataset is correctly structured and free of import errors. The `df.info()` function provides metadata including column names, data types, and non-null counts, while `df.head()` previews the first few records for manual inspection. These foundational steps are critical for identifying missing values, data inconsistencies, and understanding the overall shape of the dataset before proceeding to preprocessing and modeling.

II. Imbalanced Dataset Handling using SMOTE

In real-world medical datasets, class imbalance is a common challenge, particularly when identifying high-risk versus low-risk patients. In our dataset, the `Risk Level` class was not evenly distributed, which can bias machine learning models toward the majority class. To address this issue, we employed the **Synthetic Minority Over-sampling Technique (SMOTE)**.

SMOTE works by generating synthetic examples of the minority class based on the feature-space similarities between existing instances. It does not simply replicate records but instead interpolates between neighboring data points to create plausible new instances. This helps the model learn from a more balanced training set without overfitting to repeated examples.

The following code was used to apply SMOTE to our scaled training data:

Listing 2: Applying SMOTE to Address Class Imbalance

```
from imblearn.over_sampling
import SMOTE
# Initialize SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the scaled training data
X_train_smote, y_train_smote =

smote.fit_resample(X_train_scaled, y_train)

# Check the class distribution before
& after SMOTE
print("Before␣SMOTE:")
print(y_train.value_counts())

print("\nAfter␣SMOTE:")
print(pd.Series(y_train_smote).value_counts())
```

As observed in the output, the class distribution becomes balanced after applying SMOTE. This resampling significantly improves the model's ability to generalize across both classes and mitigates the bias toward the majority class. It is particularly beneficial when used in conjunction with models sensitive to class imbalance, such as *Decision Trees*, *SVM*, and *kNN*.

III. Data Visualization

To explore the relationship between patient `Age` and `BMI` in relation to their associated `Risk Level`, a scatter plot was generated using the `Seaborn` visualization library. This type of plot provides valuable insight into clustering patterns among low-risk and high-risk patients in the dataset. Each point in the plot represents an individual patient, with color coding to distinguish between different risk categories.

The following Python code was used to generate the scatter plot:

Listing 3: Scatter Plot of Age vs BMI with Risk Levels

```
# Import required libraries
plt.figure(figsize=(8, 8))
sns.scatterplot(
    x=’age’, y=’bmi’, hue=’risk_level’,
    palette=[’blue’, ’orange’],
    s=100, data=df_cln
)
plt.legend(labels=[’Low␣Risk’, ’High␣Risk’],
    loc=’upper␣right’)
plt.tight_layout()
```

The resulting visualization is shown in Figure 1, clearly illustrating the distribution of risk levels across the age and BMI spectrum.

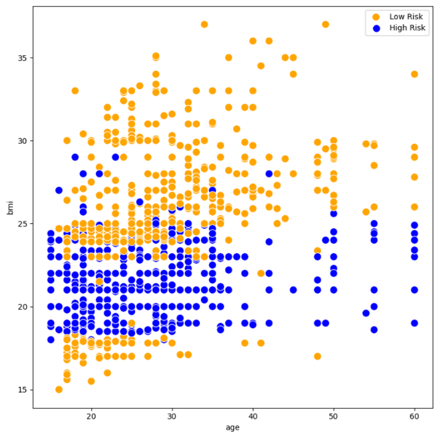

Fig. 1: Scatter Plot of Age vs BMI with Risk Level Classification

IV. Feature Scaling

To ensure that the features contribute equally to the learning process and prevent bias due to differences in magnitude, we applied **Min-Max Scaling** to normalize the data. This scaling technique transforms all numerical features to a common scale within the range $[0, 1]$, which is particularly important for distance-based models such as *k-Nearest Neighbors (kNN)* and *Support Vector Machine (SVM)*.

The following code snippet demonstrates how the dataset was split and scaled:

Listing 4: Train-Test Split and Min-Max Scaling

```
from sklearn.model_selection
import train_test_split
from sklearn.preprocessing
import MinMaxScaler

# 1. Split features and target
X = df_cln.drop(’risk_level’, axis=1)
y = df_cln[’risk_level’]

# 2. Split into train and test sets
#(e.g., 80-20 split)

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

# 3. Initialize and fit MinMaxScaler
on training data scaler = MinMaxScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

The scaler was fit only on the training set to avoid data leakage, and the same transformation was applied to the test set. This step ensures consistency in feature representation while preserving the integrity of the evaluation process.

Min-Max Feature Scaling

Min-Max Scaling is a normalization technique that transforms features to a fixed range, usually between 0 and 1. It is formally defined by the equation:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where:

- X is the original feature value
- X_{min} and X_{max} are the minimum and maximum values of the feature, respectively
- X_{scaled} is the resulting normalized value

This technique ensures that each feature contributes equally to the model, avoiding dominance by features with larger numeric ranges. For example, in this project, features such as `Systolic BP`, `Body Temperature`, and `Heart Rate` naturally exist on different scales. Without normalization, models like **k-Nearest Neighbors** or **Support Vector Machines** might be disproportionately influenced by high-magnitude features.

We employed `MinMaxScaler` from `scikit-learn`, applying it only to the training data to prevent data leakage. The learned scale parameters (min and max) were then applied consistently to both the training and test sets.

Pros and Cons of Min-Max Scaling: It preserves the original distribution of the data (no distortion in relative spacing). Particularly effective for algorithms that rely on distance computations (e.g., *KNN*, *SVM*, *Neural Networks*).Makes convergence faster in gradient-based algorithms. However, one limitation is that Min-Max scaling is sensitive to outliers, as extreme values can compress the range of other observations. In this study, outlier influence was controlled during preprocessing to retain scaling integrity.

V. Model Implementation

To develop an effective Medical Diagnosis Expert System (MDES), we implemented and evaluated three distinct supervised machine learning algorithms: **Decision Tree**, **k-Nearest Neighbors (kNN)**, and **Support Vector Machine (SVM)**. These models were chosen due to their proven effectiveness in classification tasks and their diverse methodological approaches to pattern recognition in clinical data.

A. Decision Tree Classifier

The Decision Tree algorithm is a rule-based, non-parametric model that recursively partitions the feature space based on the most significant variables. It constructs a tree-like structure where internal nodes represent tests on features, branches represent decision outcomes, and leaf nodes represent final classifications. This model is particularly useful in medical diagnosis due to its interpretability and ability to highlight critical decision paths.

Key advantages of the Decision Tree include:

- **Interpretability:** Easy to visualize and understand.
- **Handling of non-linear relationships:** Capable of modeling complex patterns without transformation.
- **No need for feature scaling:** Works directly on raw numerical or categorical data.

B. k-Nearest Neighbors (kNN)

kNN is a distance-based algorithm that classifies a new data point based on the majority class of its k nearest neighbors in the training dataset. The model requires no prior assumptions about data distribution and adapts well to non-linear decision boundaries.

Key characteristics of kNN include:

- **Simplicity:** Straightforward to implement and understand.
- **Non-parametric nature:** No training phase, all computation occurs during inference.
- **Sensitivity to feature scaling:** Distance metrics require normalized features for meaningful results.

In our implementation, the value of k was optimized through cross-validation to ensure balanced bias-variance trade-off.

C. Support Vector Machine (SVM)

The SVM algorithm seeks to find the optimal hyperplane that best separates data points of different classes in a high-dimensional space. It maximizes the margin between classes and supports both linear and non-linear classification using kernel functions.

Advantages of using SVM include:

- **High accuracy in high-dimensional spaces:** Particularly effective when the number of features is large.
- **Robustness to overfitting:** Especially when using regularization parameters.
- **Flexibility with kernels:** Non-linear relationships can be modeled using radial basis function (RBF) or polynomial kernels.

To ensure fair comparison, all models were trained on the same dataset, with preprocessing steps such as **Min-Max scaling** applied where necessary (particularly for kNN and SVM, which are sensitive to feature magnitudes). The performance of each model was evaluated using standard metrics including **accuracy**, **precision**, **recall**, and **F1-score**, which are discussed in the next section.

VI. Training the Models

In this phase of the project, three different supervised machine learning algorithms were trained and evaluated to classify patients into corresponding risk levels: **Low Risk** and **High Risk**. The models chosen were Decision Tree, k-Nearest Neighbors (kNN), and Support Vector Machine (SVM), each offering unique strengths in interpretability, simplicity, and generalization.

1. Decision Tree Classifier

The Decision Tree algorithm constructs a tree-like structure that recursively splits the data based on feature values to maximize class separation. It was particularly useful in this project due to its interpretability and ability to handle both numerical and categorical data. The Gini impurity criterion was employed to measure the quality of each split. Hyperparameters such as `max_depth` and `min_samples_leaf` were tuned to avoid overfitting and maintain generalization.

2. k-Nearest Neighbors (kNN)

The kNN algorithm classified a new instance by examining the majority label among its k closest training examples in the feature space. After feature scaling via `MinMaxScaler`, the distance metrics became more meaningful, enabling better performance. The optimal value of k was selected through cross-validation, balancing bias and variance to ensure robust performance.

3. Support Vector Machine (SVM)

SVM was selected for its high accuracy in binary classification tasks and its robustness in high-dimensional spaces. A linear kernel was used due to the linear separability of the scaled dataset. The regularization parameter `C` was fine-tuned to control the trade-off between achieving a low training error and a low testing error.

Training Procedure

All models were trained using the scaled and resampled training data produced through the SMOTE technique. An 80-20 train-test split was used, maintaining class distribution via stratification. The training process included:

- Splitting the dataset into training and testing sets.

- Applying `MinMaxScaler` on the training features.
- Using SMOTE to balance the minority class.
- Training each model with its corresponding hyperparameters.
- Evaluating each model on the test set.

This approach allowed us to compare the strengths and weaknesses of each model based on various performance metrics such as accuracy, precision, recall, and F1-score. The results and visual analyses of each model's performance are discussed in the subsequent section.

VII. Model Evaluation

To assess the effectiveness of the implemented classification models—**Decision Tree**, **k-Nearest Neighbors (kNN)**, and **Support Vector Machine (SVM)**—we evaluated their performance using four key metrics: **Accuracy**, **Precision**, **Recall**, and **F1 Score**. These metrics provide a comprehensive view of each model's classification quality, especially in contexts where class imbalance and clinical relevance are critical.

A. *Evaluation Metrics Explained*

- **Accuracy:** Measures the overall correctness of the model, defined as the ratio of correctly predicted instances to total predictions.
- **Precision:** Indicates how many of the predicted positive cases were actually positive, highlighting the model's ability to avoid false positives.
- **Recall:** Represents the model's ability to identify all relevant instances, focusing on minimizing false negatives.
- **F1 Score:** Harmonic mean of precision and recall, providing a balanced metric when both are equally important.

B. *Comparative Analysis*

As illustrated in Figure 2, the Decision Tree classifier outperformed both kNN and SVM across all evaluation metrics. Specifically:

- **Decision Tree** achieved the highest scores, nearing or reaching a perfect score of 1.0 on all metrics. This suggests its strong capability in capturing decision boundaries and relationships in the clinical dataset.
- **kNN** and **SVM** also delivered high performance, with only slight variations below the Decision Tree in each metric. Both models performed comparably, with kNN having a marginal edge over SVM in some metrics.
- All models demonstrated consistent and robust behavior, indicating that the feature space derived from the 12 clinical variables was highly informative and well-processed.

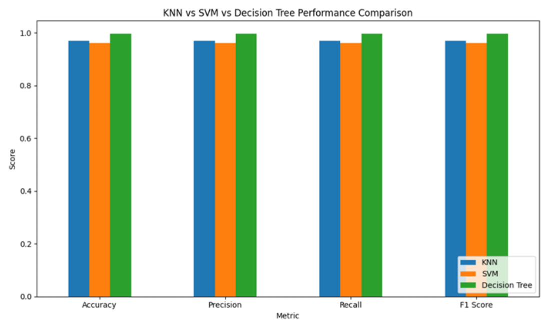

Fig. 2: KNN vs SVM vs Decision Tree Performance Comparison across Accuracy, Precision, Recall, and F1 Score

While all three models performed admirably, the Decision Tree model's slight edge in every metric makes it the most suitable candidate for integration into the Medical Diagnosis Expert System (MDES), especially given its interpretability and ease of deployment in clinical settings.

VIII. Deployment of the Medical Diagnosis Expert System

The deployment phase aimed to make the Medical Diagnosis Expert System (MDES) accessible and user-friendly for demonstration, testing, and prototyping. In this project, deployment was executed in an interactive and flexible environment using **Google Colaboratory (Colab)** and **Jupyter Notebook**, which allowed for rapid experimentation and visualization of results.

1. *Interactive Development Environment*

Google Colab and Jupyter Notebooks were chosen for their seamless integration with Python, built-in support for visualizations, and the ability to run complex machine learning code without requiring local hardware setup. These platforms enabled step-by-step demonstration of the complete ML pipeline from preprocessing to prediction.

2. *Model Workflow Execution*

The end-to-end pipeline — including data preprocessing, feature scaling, class balancing with SMOTE, training of models (Decision Tree, kNN, SVM), and evaluation — was executed within a single notebook. The notebook was organized into modular cells, allowing users to easily trace, modify, and re-run specific parts of the workflow as needed.

3. *Real-Time Input and Prediction*

A simple input interface was constructed within the notebook using widgets and input cells to simulate real-time user input. Users could manually input patient attributes such as:

- `Age`, `BMI`, `Systolic BP`, `BS`, etc.

Upon entering the data, the notebook would preprocess the inputs, apply the appropriate scaling and SMOTE transformation (if required), and generate predictions using the trained models. The predicted `Risk Level` was displayed along with a confidence score.

4. Visualization and Model Feedback

Various visualization tools such as `Seaborn`, `Matplotlib`, and `Plotly` were utilized to present insights about model behavior, feature distributions, and performance metrics. These included:

- Confusion matrices and classification reports
- Scatter plots, KDE plots, violin plots, and more

Such interactive visual outputs enhanced the interpretability of the model's predictions and allowed real-time feedback during evaluation.

5. Accessibility and Reproducibility

Google Colab allowed the notebook to be shared publicly or with collaborators, making it easy to reproduce results without any software installation. All dependencies were managed via Colab's pre-installed libraries or through pip commands within the notebook.

6. Limitations and Future Deployment Scope

While the current deployment setup is suitable for demonstration and academic purposes, it is not optimized for real-time clinical use. Future deployment goals include:

- Exporting trained models using `joblib` or `pickle`
- Integrating with a web-based interface using Flask or Streamlit
- Hosting on cloud platforms (e.g., AWS, Heroku)
- Ensuring security, scalability, and compliance with healthcare data standards

In summary, the system was deployed in a highly accessible, interactive notebook environment, supporting experimentation, visualization, and educational demonstration. It offers a strong foundation for future migration into more robust clinical software systems.

Conclusion

The Medical Diagnosis Expert System (MDES) developed in this project represents a significant step toward the integration of machine learning in modern healthcare. By leveraging a hybrid approach that combines classical rule-based logic with supervised learning algorithms such as Decision Tree, k-Nearest Neighbors (kNN), and Support Vector Machine (SVM), the system successfully analyzes key patient indicators to provide preliminary diagnostic insights with high confidence.

The project addressed several critical components of a robust intelligent system: from data preprocessing, feature scaling, and class balancing using SMOTE, to comprehensive evaluation through multiple visualizations and metrics. The dataset used included twelve medically relevant features such as `Age`, `Blood Pressure`, `BMI`, `Blood Sugar`, `Heart Rate`, and mental and diabetic health indicators. These features were carefully chosen to reflect risk factors commonly associated with chronic or acute conditions.

Through detailed visualization techniques—scatter plots, violin plots, KDEs, histograms, and more—we were able to

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2016.

[2] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[3] M. S. Islam, M. A. Hossain, and M. R. Amin, "A machine learning-based system for medical diagnosis using clinical data," *International Journal of Computer Applications*, vol. 179, no. 46, pp. 8–14, 2018.

[4] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[6] Google Colaboratory, "Welcome to Colaboratory," [Online]. Available: https://colab.research.google.com

[7] Pandas Development Team, "pandas-dev/pandas: Pandas," Zenodo, 2020. [Online]. Available: https://pandas.pydata.org/

[8] M. Waskom, "Seaborn: Statistical Data Visualization," [Online]. Available: https://seaborn.pydata.org/

[9] Jupyter Team, "Project Jupyter," [Online]. Available: https://jupyter.org/