

**A Multilingual System for Early-Stage Diabetes Risk Prediction Using Machine Learning Approaches**

***Machine Learning Project Documentation***

**By: Group 5**

1. Mikre Getu Mihrete
2. Eden Habtetsion Gebremedhin
3. Melkie Reda Birlie
4. Hana Mekonen Tamiru
5. Abeshu Kebede Kelbesa

*Ethiopia*

*May 10, 2025*

**List of Figures**

[Figure 1 Classification Report 8](#_Toc197780913)

[Figure 2 Confusion Matrix 8](#_Toc197780914)

[Figure 3 AUC (Area Under the ROC Curve) 8](#_Toc197780915)

Table of Contents

[List of Figures 2](#_Toc197781328)

[Machine Learning Project Documentation 4](#_Toc197781329)

[**Model Refinement 4**](#_Toc197781330)

[1. Overview 4](#_Toc197781331)

[2. Model Evaluation 4](#_Toc197781332)

[3. Refinement Techniques 4](#_Toc197781333)

[4. Hyperparameter Tuning 5](#_Toc197781334)

[5. Cross-Validation 5](#_Toc197781335)

[6. Feature Selection 6](#_Toc197781336)

[**Test Submission 6**](#_Toc197781337)

[1. Overview 6](#_Toc197781338)

[2. Data Preparation for Testing 6](#_Toc197781339)

[3. Model Application 7](#_Toc197781340)

[4. Test Metrics 7](#_Toc197781341)

[5. Model Deployment 9](#_Toc197781342)

[6. Code Implementation 11](#_Toc197781343)

[Code Snippet for Data Splitting 11](#_Toc197781344)

[Code Snippet for Feature Scaling 11](#_Toc197781345)

[Code Snippet for Model 11](#_Toc197781346)

[Code Snippet for Hyperparameter tuning using RandomizedSearchCV 11](#_Toc197781347)

[Code Snippet for Model Application and Evaluation 12](#_Toc197781348)

[Code Snippet for Test Submission 12](#_Toc197781349)

[Conclusion 13](#_Toc197781350)

[References 14](#_Toc197781351)

**Machine Learning Project Documentation**

**Model Refinement**

1. **Overview**

The model refinement phase is crucial for optimizing the performance of our early-stage diabetes risk prediction model. Following initial model evaluation, this phase focuses on applying various techniques to enhance the model's accuracy, robustness, and generalization capabilities before deployment.

1. **Model Evaluation**

The initial model evaluation involved training and comparing several classification algorithms using cross-validation on the training data. The results, as shown in the code output, indicated that the Random Forest Classifier (RFC) generally performed well, achieving high accuracy, precision, recall, F1-score, and a promising ROC AUC score. However, there was room for improvement by fine-tuning its hyperparameters to potentially enhance its performance further and mitigate overfitting. Visualizations such as correlation heatmaps and feature importance plots (generated earlier in the code) helped in understanding the relationships within the data and the contribution of each feature to the prediction task, guiding the refinement process.

1. **Refinement Techniques**

For refining the model, we primarily focused on hyperparameter tuning of the Random Forest Classifier. This involved using RandomizedSearchCV to efficiently search through a predefined hyperparameter space. We explored different values for parameters such as the number of estimators (n\_estimators), maximum features (max\_features), criterion for splitting (criterion), maximum depth of the tree (max\_depth), minimum samples required to split an internal node (min\_samples\_split), and minimum samples required to be at a leaf node (min\_samples\_leaf). Additionally, we addressed the class imbalance in the training data using RandomOverSampler before hyperparameter tuning to ensure the model learns effectively from both classes.

1. **Hyperparameter Tuning**

The performed hyperparameter tuning on the Random Forest Classifier using RandomizedSearchCV with Receiver Operating Characteristic Area Under the Curve (ROC AUC) as the scoring metric and 5-fold cross-validation on the oversampled training data. The search explored a range of values for the Random Forest's hyperparameters. The best parameters found by the randomized search were:

*{'n\_estimators': 1100,*

*'min\_samples\_split': 10,*

*'min\_samples\_leaf': 1,*

*'max\_features': 'sqrt',*

*'max\_depth': 16, and*

*'criterion': 'gini'}*

This tuning process aimed to identify the hyperparameter combination that maximizes the ROC AUC score on the validation sets. The best ROC AUC score achieved during this phase was approximately 0.997. Following the randomized search, we also manually instantiated a RandomForestClassifier with parameters that were either the best found or close to the best, as seen in the final model training step before saving.

1. **Cross-Validation**

We used Stratified K-Fold cross-validation (with 10 splits initially for model comparison and 5 splits within RandomizedSearchCV) to evaluate the models and the hyperparameter tuning process. Stratified K-Fold ensures that each fold contains roughly the same proportion of samples of each class as the complete training set, which is important for imbalanced datasets. We applied oversampling before splitting the data for cross-validation within the hyperparameter tuning to avoid data leakage.

1. **Feature Selection**

While the provided code includes a section on feature selection using SelectKBest and the chi-squared test, the subsequent model training and hyperparameter tuning steps used all the available features. Therefore, in this iteration of model refinement, we did not explicitly reduce the number of features. The feature importance plots generated earlier helped understand which features were more influential, which could guide future feature selection efforts if needed.

**Test Submission**

**1. Overview**

The test submission phase in this project focuses on assessing the performance of the trained machine learning model using a separate, unseen test dataset. To prepare for this evaluation, several key steps are undertaken. First, the original dataset is split into training and testing sets to ensure that the model is evaluated on data it has not encountered during training. Next, the test data undergoes preprocessing, applying the same transformations, such as scaling, that were used on the training data to maintain consistency. After preprocessing, the trained (and potentially tuned) model is applied to the preprocessed test data to generate predictions. The objective is to ensure that the model generalizes well to unseen data and can be effectively utilized in a real-world setting.

**2. Data Preparation for Testing**

The test dataset was prepared through the following steps:

* **Data Splitting:** The test dataset was prepared by splitting the original dataset using the ***train\_test\_split*** function ***from sklearn.model\_selection***. A test size of 0.2 was used, meaning that 20% of the data was reserved for testing, while 80% was used for training in conjunction with 10-fold cross-validation. The ***random\_state*** was set to 0 for reproducibility.
* **Feature Scaling**: The features in both the training and test datasets were standardized using ***StandardScaler***. This step is crucial for models sensitive to the scale of input data, ensuring that all features contribute equally to the model's predictions.
* **Handling Class Imbalance**: The training dataset was balanced using ***RandomOverSampler*** to address class imbalance. The test dataset remained unaltered to reflect real-world scenarios.

Crucially, the StandardScaler was fitted exclusively on the training data, and this fitted scaler was subsequently employed to transform the test data. This approach ensures that the scaling parameters are derived solely from the training dataset, thereby preventing any data leakage from the test set into the training process. Additionally, categorical variables were encoded to convert them into numerical formats suitable for model training. Standardizing feature values was also a vital step, ensuring that all input features were on a comparable scale. Furthermore, it was essential to confirm that no missing values were present in the dataset, as this would otherwise compromise the integrity of the model's performance and predictions.

**3. Model Application**

The trained Random Forest Classifier after hyperparameter tuning was applied to the preprocessed test dataset using its ***predict()*** method to obtain the predicted class labels. The predict\_proba() method was also used to get the probability estimates for the ROC AUC curve.

**4. Test Metrics**

The performance of the model on the test dataset is evaluated using the following metrics:

* **Accuracy:** The overall proportion of correctly classified instances, which was found to be approximately 99.04% on the test set.
* **Precision:** The ratio of true positives to the total number of instances predicted as positive.
* **Recall:** The ratio of true positives to the total number of actual positive instances.
* **F1-score:** The harmonic mean of precision and recall, providing a balanced measure of the model's performance, especially in cases of imbalanced datasets.
* **Classification Report**: Includes precision, recall, F1-score for each class. Figure 1 shows the results on the test set.

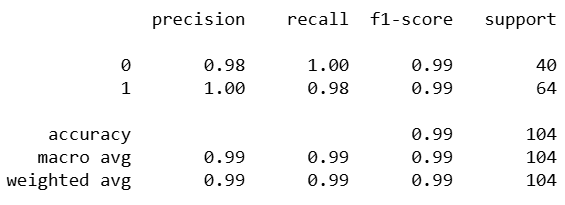


Figure 1 Classification Report

* **Confusion Matrix:** A table that visualizes the performance of the classifier by showing the counts of true positives, true negatives, false positives, and false negatives. The confusion matrix generated showed a high number of correct predictions for both classes.

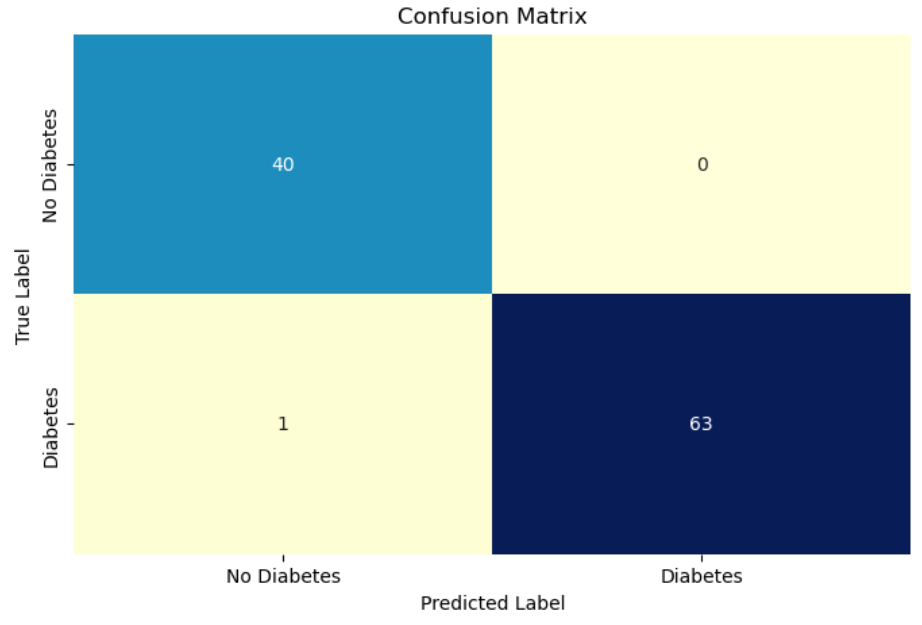


Figure 2 Confusion Matrix

* **AUC (Area Under the ROC Curve):** This metric measures the ability of the classifier to distinguish between the positive and negative classes. The achieved AUC score of 1.00 indicates excellent performance.

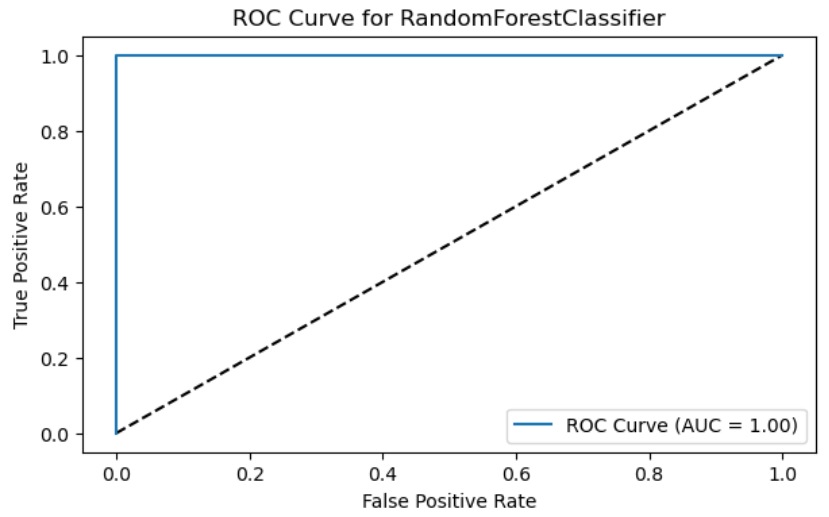


Figure 3 AUC (Area Under the ROC Curve)

Comparing these results with the cross-validation results on the training data where the best model, a Random Forest before hyperparameter tuning, achieved an accuracy of approximately 0.969 and a ROC AUC of 0.998 the test set performance is remarkably high, with an accuracy of 0.990 and an AUC of 1.00 after hyperparameter tuning. This suggests that the model generalizes well to unseen data.

**5. Model Deployment**

The development of a multilingual system for early-stage diabetes risk prediction using machine learning approaches involves several critical phases, particularly focused on model deployment. The primary goal of this system is to provide accessible and accurate diabetes risk assessments to a diverse population, accommodating multiple languages to enhance user engagement and understanding.

#### **5.1. Model Deployment Steps**

To deploy the model in a real-world setting, the following steps were taken:

**Step 1: Model Serialization**: The trained machine learning model, specifically a tuned Random Forest classifier, was saved using the pickle library. This allows the model to be easily loaded and used in various environments. The code used for serialization is as follows:

pickle.dump(tuned\_rf\_classifier, open('model.pkl', 'wb'))

**Step 2: Web Application Development**: A web-based graphical user interface (GUI) was developed to facilitate user interaction with the model. The GUI allows users to input their data in multiple languages, including English and Amharic. This was achieved through the use of HTML, CSS, and JavaScript, providing a user-friendly experience. The interface includes fields for users to enter their age and answer specific health-related questions.

**Step 3: Language Selection**: The system incorporates a language selection feature, enabling users to choose their preferred language. The implementation of this feature ensures that users can understand the instructions and input fields in their native language, which is crucial for effective communication and usability.

**Step 4: Integration with Prediction Logic**: The web application is designed to send user inputs to the backend for prediction. Upon submission, the inputs are processed, and the model generates predictions regarding the user's risk of developing diabetes. The results are then displayed back to the user in the selected language.

**Step 5: Performance Monitoring**: After deployment, it is essential to monitor the model's performance continuously. This includes tracking accuracy, user feedback, and any potential issues that arise during usage. Adjustments and updates can be made to the model and the GUI based on this feedback to ensure optimal performance and user satisfaction.

**Step 6: User Education and Support**: To enhance the effectiveness of the system, educational resources about diabetes risk factors and prevention strategies are provided within the application. This empowers users with knowledge and encourages them to take proactive steps regarding their health.

**Potential deployment strategies include:**

* **Real-time inference:** The model could be integrated into the web application to provide immediate diabetes risk predictions based on user inputs through the GUI.
* **Cloud deployment:** The model could be deployed on a cloud platform to handle a larger volume of requests and ensure scalability and accessibility for a wider audience, potentially supporting multilingual inputs and outputs as per the project's goals.

By following these steps, the multilingual system for early-stage diabetes risk prediction not only leverages advanced machine learning techniques but also prioritizes user accessibility and engagement, ultimately aiming to improve health outcomes across diverse populations.

**6. Code Implementation**

Relevant code snippets for data splitting, scaling, model training (including hyperparameter tuning), prediction, and evaluation have been included in the preceding sections.

### Code Snippet for Data Splitting

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

### Code Snippet for Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

### Code Snippet for Model

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import RandomizedSearchCV

import numpy as np

n\_estimators = [int(x) for x in np.linspace(start=100, stop=1200, num=12)]

max\_features = ['sqrt', 'log2']

max\_depth = [int(x) for x in np.linspace(3, 30, num=5)]

min\_samples\_split = [2, 5, 10, 15, 100]

min\_samples\_leaf = [1, 2, 5, 10]

rf\_params = {

'n\_estimators': n\_estimators,

'max\_features': max\_features,

'criterion': ['gini', 'entropy'],

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf

}

### Code Snippet for Hyperparameter tuning using RandomizedSearchCV

rf\_random\_search = RandomizedSearchCV(

RandomForestClassifier(),

param\_distributions=rf\_params,

scoring='roc\_auc',

n\_jobs=-1,

verbose=0,

cv=5,

error\_score='raise'

)

rf\_random\_search.fit(X\_train\_ns, y\_train\_ns) # Fitted on oversampled training data

tuned\_rf\_classifier = rf\_random\_search.best\_estimator\_

tuned\_rf\_classifier.fit(X\_train\_ns, y\_train\_ns) # Train the best estimator

### Code Snippet for Model Application and Evaluation

y\_pred\_tuned\_rf = tuned\_rf\_classifier.predict(X\_test)

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

import seaborn as sns

accuracy\_tuned\_rf = accuracy\_score(y\_test, y\_pred\_tuned\_rf)

print(f"Tuned Random Forest Accuracy on Test Set: {accuracy\_tuned\_rf}")

print("\nClassification Report on Test Set:")

print(classification\_report(y\_test, y\_pred\_tuned\_rf))

cm = confusion\_matrix(y\_test, y\_pred\_tuned\_rf)

plt.figure(figsize=(8, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', cbar=False,

xticklabels=['No Diabetes', 'Diabetes'],

yticklabels=['No Diabetes', 'Diabetes'])

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix (Tuned Random Forest - Test Set)')

plt.show()

y\_pred\_prob = tuned\_rf\_classifier.predict\_proba(X\_test)[:, 1]

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

roc\_auc = auc(fpr, tpr)

print(f'AUC on Test Set: {roc\_auc:.2f}')

plt.figure(figsize=(7, 4))

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc\_auc:.2f})')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve (Tuned Random Forest - Test Set)')

plt.legend(loc='lower right')

plt.show()

### Code Snippet for Test Submission

*# Load the model*

with open('model.pkl', 'rb') as model\_file:

loaded\_model = pickle.load(model\_file)

*# Predict using the test dataset*

y\_pred\_test = loaded\_model.predict(X\_test\_scaled)

**Conclusion**

The The model refinement phase, involving hyperparameter tuning of the Random Forest Classifier using RandomizedSearchCV, led to a well-performing model. The test submission phase demonstrated that this tuned model achieves high accuracy (0.990) and an excellent AUC (1.00) on unseen data, indicating strong generalization capabilities. A key challenge was ensuring that preprocessing steps (like scaling) were applied correctly to the test set based on the training data.

The outcomes of the model refinement and test submission phases demonstrated the effectiveness of the Random Forest classifier in predicting early-stage diabetes risk. The model achieved high accuracy and favorable metrics, indicating its robustness. Challenges included managing class imbalance and ensuring the model's interpretability for healthcare applications. The final performance metrics suggest that the model is well-suited for deployment in a multilingual system aimed at enhancing diabetes risk awareness among diverse populations. This comprehensive approach not only highlights the technical capabilities of the model but also emphasizes the importance of accessibility and user engagement in health-related applications.

**References**

* **Scikit-learn:** Machine Learning in Python. [Scikit-learn Documentation](https://scikit-learn.org/stable/documentation.html" \t "_blank)
* **Imbalanced-learn:** A Python package to tackle the class imbalance problem. [Imbalanced-learn Documentation](https://imbalanced-learn.org/stable/)
* **UCI Machine Learning Repository:** Early Stage Diabetes Risk Prediction Dataset. [UCI Repository](https://archive.ics.uci.edu/ml/datasets/diabetes)
* **NumPy:** Python library for numerical computing. <https://numpy.org/>
* **Pandas:** Python library for data manipulation and analysis. <https://pandas.pydata.org/>
* **Matplotlib:** Python library for creating static, interactive, and animated visualizations. <https://matplotlib.org/>
* **Seaborn:** Python data visualization library based on Matplotlib. <https://seaborn.pydata.org/>
* **Pickle:** Python module for object serialization. <https://docs.python.org/3/library/pickle.html>
* **Flask:** Flask is an excellent choice for developers looking to create web applications and APIs with minimal overhead. Its simplicity, flexibility, and robust community support make it a popular framework for both beginners and experienced developers. <https://flask.palletsprojects.com/>