ABSTRACT

Disease Prediction using ML

"Harnessing the power of data and algorithms to predict health outcomes for a safer tomorrow."

This project focuses on building a robust disease prediction system using machine learning techniques. By utilizing a comprehensive dataset containing symptom and disease relationships, the model is designed to predict diseases based on the symptoms provided by the user. The dataset, consisting of 132 symptom features and a target disease column, is pre-processed, cleaned, and converted into a suitable format for model training. Three classifiers—Support Vector Classifier (SVC), Naive Bayes, and Random Forest—are employed to build predictive models. To enhance prediction accuracy, the outputs of these classifiers are combined through a voting mechanism, ensuring that even if one model errs, the others can correct it.

The implementation involves data splitting for training and testing, cross-validation for model evaluation, and the use of confusion matrices to assess model performance. The final model is trained on the entire dataset and is capable of providing disease predictions based on user-input symptoms. This approach not only demonstrates the practical application of machine learning in healthcare but also offers the potential for early diagnosis and proactive disease management. The system provides a user-friendly interface to input symptoms and receive disease predictions, making it an essential tool in supporting healthcare professionals and patients alike.

Introduction

Overview of the Project

In the era of artificial intelligence and machine learning, healthcare systems are rapidly adopting these technologies to improve diagnostic accuracy and patient outcomes. One critical area where machine learning can play a transformative role is in disease diagnosis. Early and accurate disease prediction is essential for effective treatment and management.

This project focuses on building a machine-learning model that predicts diseases based on the symptoms a person experiences. By leveraging various machine learning algorithms, such as Support Vector Classifier (SVC), Naive Bayes (NB), and Random Forest (RF), this system aims to predict a person's disease based on the input symptoms. The model utilizes a dataset where symptoms are represented as binary features, and the target variable, prognosis, is the disease the person is suffering from.

The dataset, available from Kaggle, consists of 133 columns: 132 columns representing symptoms (in binary format) and the last column representing the disease type. The model employs a combination of multiple classifiers to make robust predictions, ensuring more accurate and reliable results by reducing the likelihood of errors from individual models.

Proposed System

The key features of the Disease Prediction System using Machine Learning are:

1. **Disease Prediction Based on Symptoms:**
   - The system allows users to input a set of symptoms and predicts the most likely disease(s) based on the symptoms entered. The prediction is based on a trained machine learning model that uses historical data to map symptoms to diseases.

2. **Multiple Classifier Integration:**
   - The system combines predictions from three different classifiers: Support Vector Classifier (SVC), Naive Bayes (NB), and Random Forest (RF). By combining results from multiple models, the system improves prediction accuracy and reduces error.

3. **High Accuracy:**
   - The model achieves high accuracy by using a combination of classifiers, leveraging the strengths of each one. The aggregated prediction (mode of individual classifiers) ensures that the final prediction is reliable and correct.

4. **Real-time Prediction:**
   - The system can accept user input in real time and immediately provide predictions, making it suitable for practical use in healthcare settings.

5. **User-friendly Interface:**
    - The system is designed to be simple and easy to use. Healthcare professionals or users can input symptom data in a straightforward manner (as a list or form), making the tool accessible even for non-technical users.

6. **Comprehensive Data Preprocessing:**
    - The system preprocesses the input data, including encoding symptoms and handling any missing or erroneous data. This ensures that the data fed into the model is clean and ready for accurate predictions.

7. **Performance Evaluation:**
    - The system evaluates the performance of the classifiers using various metrics, such as accuracy, confusion matrix, and cross-validation scores, helping ensure that the model is robust and performs well across different datasets.

8. **Scalability:**
    - While the current dataset is based on predefined symptoms and diseases, the system is scalable. It can be extended with more symptoms, diseases, and even additional data sources for better prediction accuracy in real-world applications.

9. **Open to Future Improvements:**
    - The system's design is modular, allowing for future enhancements like incorporating more advanced algorithms (e.g., deep learning models), adding additional factors (e.g., medical history), and improving the user interface for better usability.

These features collectively enable the Disease Prediction System to offer reliable and accurate disease predictions based on symptoms, making it a potentially useful tool for healthcare professionals and patients alike.

Objectives & Scope

Objectives:

- To develop a predictive model: Create a machine learning model that can predict diseases based on input symptoms.
- To compare classifier performance: Evaluate the performance of different classifiers, including SVM, Naive Bayes, and Random Forest.
- To improve accuracy: Combine predictions from multiple classifiers to reduce prediction errors and improve accuracy.
- To implement a real-time prediction system: Develop a function that takes symptom input and returns the predicted disease in real-time.

Scope:

- The project is limited to predicting diseases based on the symptom data provided in the dataset.
- The current system only works with the predefined set of symptoms and diseases included in the dataset. For real-world application, it would need to be expanded with more diseases and symptoms.
- The model is built and tested using the dataset available on Kaggle, and further adaptation may be required for deployment in actual healthcare systems.

Requirements Specification

Functional Requirement

- Data Input: The system should accept symptom data (in the form of a string of comma-separated symptoms).
- Data Preprocessing: The input symptom data should be processed into a binary format for prediction by the machine learning models.
- Prediction Model: The system should utilize at least three classifiers: SVC, Naive Bayes, and Random Forest to predict the disease.
- Prediction Output: The system should output the predicted disease and the individual model predictions (SVM, Naive Bayes, and Random Forest).
- Performance Metrics: The model should return performance metrics such as accuracy, confusion matrix, and cross-validation scores.

Non-Functional Requirements

- Accuracy: The model should provide high accuracy on the test dataset, with an accuracy score of 100% based on the combined predictions.
- Scalability: The system should be able to handle a larger set of symptoms and diseases, should the dataset be expanded in the future.
- Usability: The system should be user-friendly, allowing healthcare professionals or users to input symptoms and get predictions effortlessly.
- Efficiency: The system should be able to provide predictions within a reasonable time frame for real-time applications.

Implementation and Results

Approach:

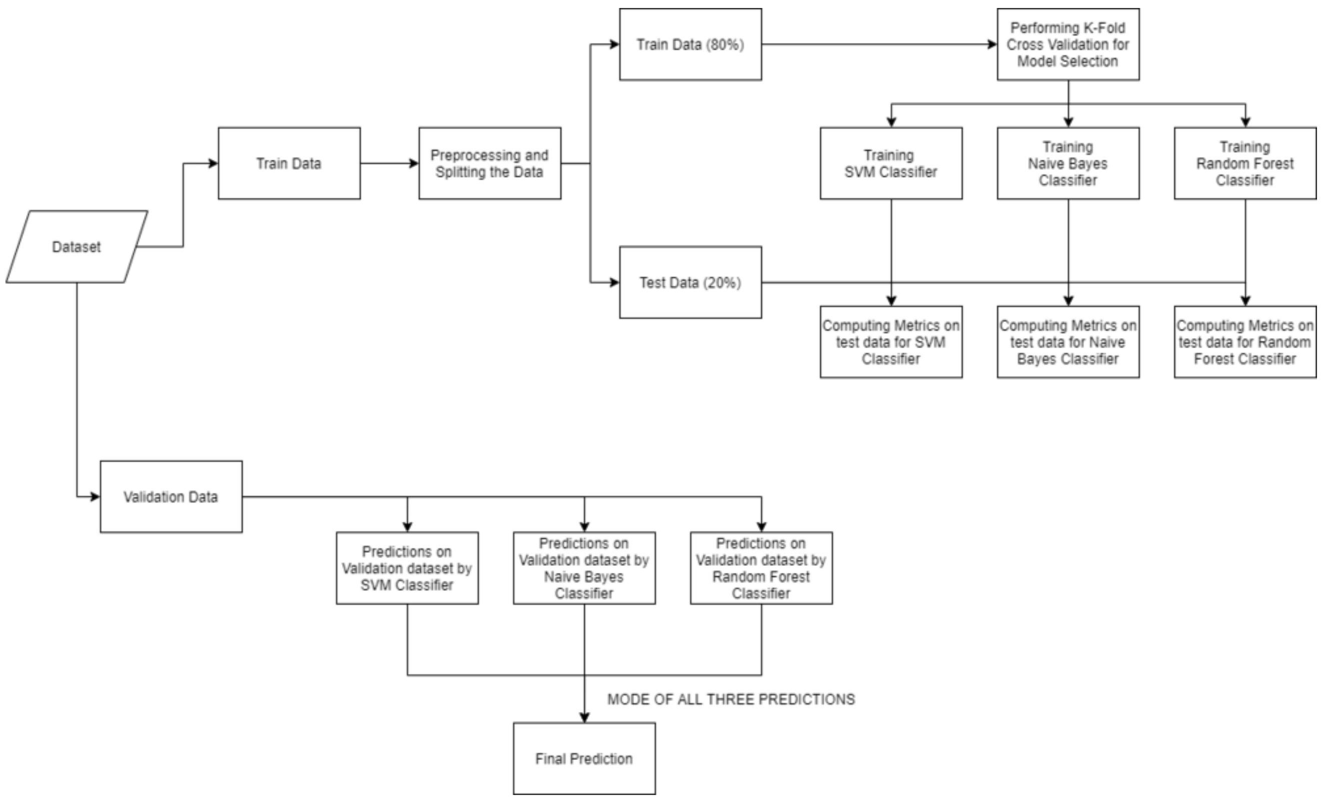
- **Gathering the Data:** Data preparation is the primary step for any machine learning problem. We will be using a dataset from Kaggle for this problem. This dataset consists of two CSV files one for training and one for testing. There is a total of 133 columns in the dataset out of which 132 columns represent the symptoms, and the last column is the prognosis.

- **Cleaning the Data:** Cleaning is the most important step in a machine learning project. The quality of our data determines the quality of our machine-learning model. So, it is always necessary to clean the data before feeding it to the model for training. In our dataset all the columns are numerical, the target column i.e. prognosis is a string type and is encoded to numerical form using a label encoder.

- **Model Building:** After gathering and cleaning the data, the data is ready and can be used to train a machine learning model. We will be using this cleaned data to train the Support Vector Classifier, Naive Bayes Classifier, and Random Forest Classifier. We will be using a confusion matrix to determine the quality of the models.

- **Inference:** After training the three models we will be predicting the disease for the input symptoms by combining the predictions of all three models. This makes our overall prediction more robust and accurate.

At last, we will be defining a function that takes symptoms separated by commas as input, predicts the disease based on the symptoms by using the trained models, and returns the predictions in a JSON format.

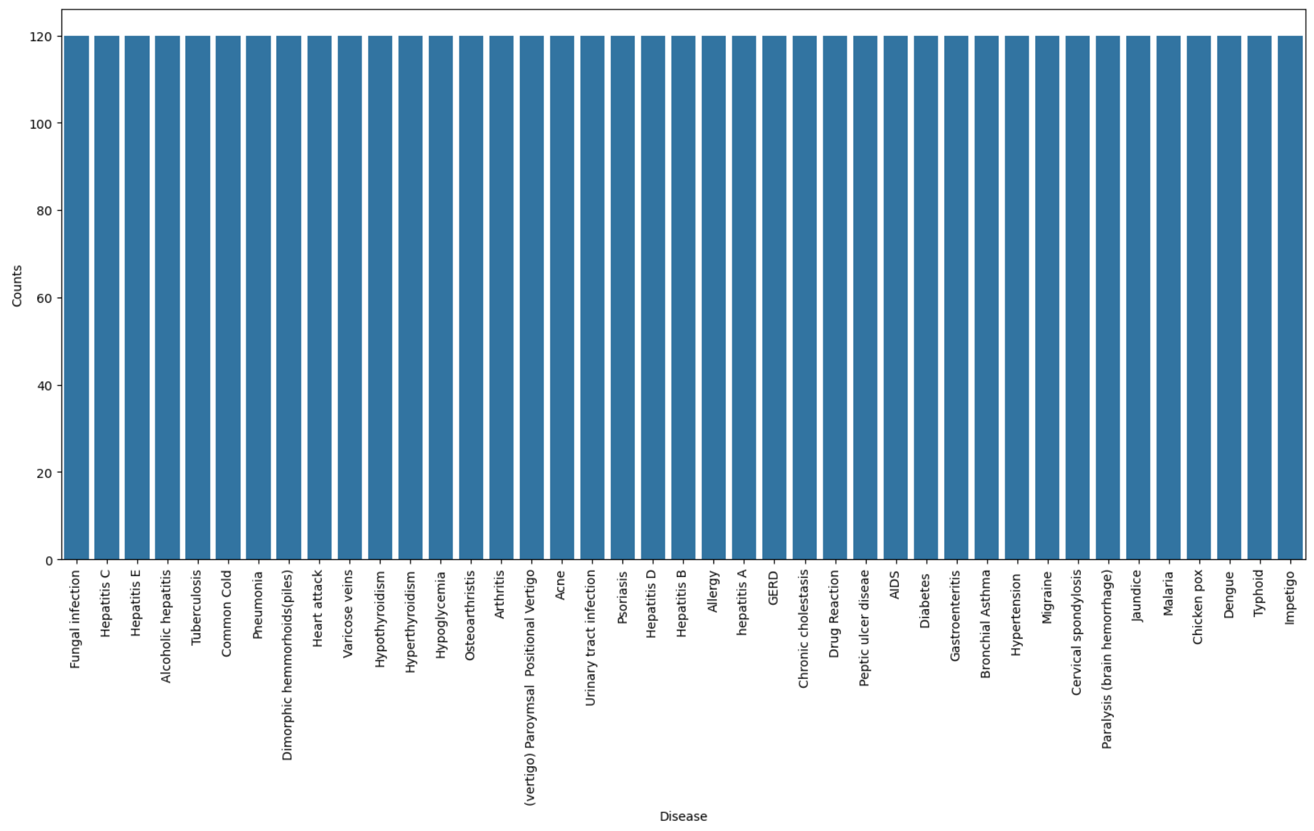


Dataset
Train Data
Preprocessing and Splitting the Data
Train Data (80%)
Test Data (20%)
Performing K-Fold Cross Validation for Model Selection
Training SVM Classifier
Training Naive Bayes Classifier
Training Random Forest Classifier
Computing Metrics on test data for SVM Classifier
Computing Metrics on test data for Naive Bayes Classifier
Computing Metrics on test data for Random Forest Classifier
Validation Data
Predictions on Validation dataset by SVM Classifier
Predictions on Validation dataset by Naive Bayes Classifier
Predictions on Validation dataset by Random Forest Classifier
MODE OF ALL THREE PREDICTIONS
Final Prediction

Reading the dataset

Firstly, we will be loading the dataset from the folders using the panda's library. While reading the dataset we will be dropping the null column. This dataset is a clean dataset with no null values and all the features consist of 0's and 1s. Whenever we are solving a classification task it is necessary to check whether our target column is balanced or not. We will be using a bar plot, to check whether the dataset is balanced or not.

```
# Reading the train.csv by removing the
# last column since it's an empty column
DATA_PATH = "dataset/Training.csv"
data = pd.read_csv(DATA_PATH).dropna(axis = 1)

# Checking whether the dataset is balanced or not
disease_counts = data["prognosis"].value_counts()
temp_df = pd.DataFrame({
    "Disease": disease_counts.index,
    "Counts": disease_counts.values
})

plt.figure(figsize = (18,8))
sns.barplot(x = "Disease", y = "Counts", data = temp_df)
plt.xticks(rotation=90)
plt.show()
```



From the above plot, we can observe that the dataset is a balanced dataset i.e. there are exactly 120 samples for each disease, and no further balancing is required. We can notice that our target column, i.e. prognosis column is of object datatype, this format is not suitable to train a machine learning model. So, we will be using a label encoder to convert the prognosis column to the numerical datatype. Label Encoder converts the labels into numerical form by assigning a unique index to the labels. If the total number of labels is n, then the numbers assigned to each label will be between 0 to n-1.

Splitting the data for training and testing the model

Now that we have cleaned our data by removing the Null values and converting the labels to numerical format, it’s time to split the data to train and test the model. We will be splitting the data into 80:20 format i.e. 80% of the dataset will be used for training the model and 20% of the data will be used to evaluate the performance of the models.

```
X = data.iloc[:,:-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test =train_test_split(
  X, y, test_size = 0.2, random_state = 24)

print(f"Train: {X_train.shape}, {y_train.shape}")
print(f"Test: {X_test.shape}, {y_test.shape}")
```

Output:

```
Train: (3936, 132), (3936,)
Test: (984, 132), (984,)
```

Model Building
After splitting the data, we will be now working on the modelling part. We will be using K-Fold cross-validation to evaluate the machine-learning models. We will be using Support Vector Classifier, Gaussian Naive Bayes Classifier, and Random Forest Classifier for cross-validation. Before moving into the implementation part let us get familiar with k-fold cross-validation and the machine learning models.

- **K-Fold Cross-Validation:** K-Fold cross-validation is one of the cross-validation techniques in which the whole dataset is split into k number of subsets, also known as folds, then training of the model is performed on the k-1 subsets and the remaining one subset is used to evaluate the model performance.
- **Support Vector Classifier:** Support Vector Classifier is a discriminative classifier i.e. when given a labelled training data, the algorithm tries to find an optimal hyperplane that accurately separates the samples into different categories in hyperspace.
- **Gaussian Naive Bayes Classifier:** It is a probabilistic machine learning algorithm that internally uses Bayes Theorem to classify the data points.
- **Random Forest Classifier:** Random Forest is an ensemble learning-based supervised machine learning classification algorithm that internally uses multiple decision trees to make the classification. In a random forest classifier, all the internal decision trees are weak learners, and the outputs of these weak decision trees are combined i.e. mode of all the predictions is as the final prediction.

Using K-Fold Cross-Validation for model selection

```
# Defining scoring metric for k-fold cross validation
def cv_scoring(estimator, X, y):
    return accuracy_score(y, estimator.predict(X))

# Initializing Models
models = {
    "SVC":SVC(),
    "Gaussian NB":GaussianNB(),
    "Random Forest":RandomForestClassifier(random_state=18)
}

# Producing cross validation score for the models
for model_name in models:
    model = models[model_name]
    scores = cross_val_score(model, X, y, cv = 10,
                             n_jobs = -1,
                             scoring = cv_scoring)
    print("=="*30)
    print(model_name)
    print(f"Scores: {scores}")
    print(f"Mean Score: {np.mean(scores)}")
```

Output:

```
============================================================
SVC
```

```
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
==================================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
==================================================================
Random Forest
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
```

From the above output, we can notice that all our machine learning algorithms are performing very well and the mean scores after k fold cross-validation are also very high. To build a robust model we can combine i.e. take the mode of the predictions of all three models so that even one of the models makes wrong predictions and the other two make correct predictions then the final output would be the correct one. This approach will help us to keep the predictions much more accurate on completely unseen data. In the below code we will be training all the three models on the train data, checking the quality of our models using a confusion matrix, and then combine the predictions of all three models.

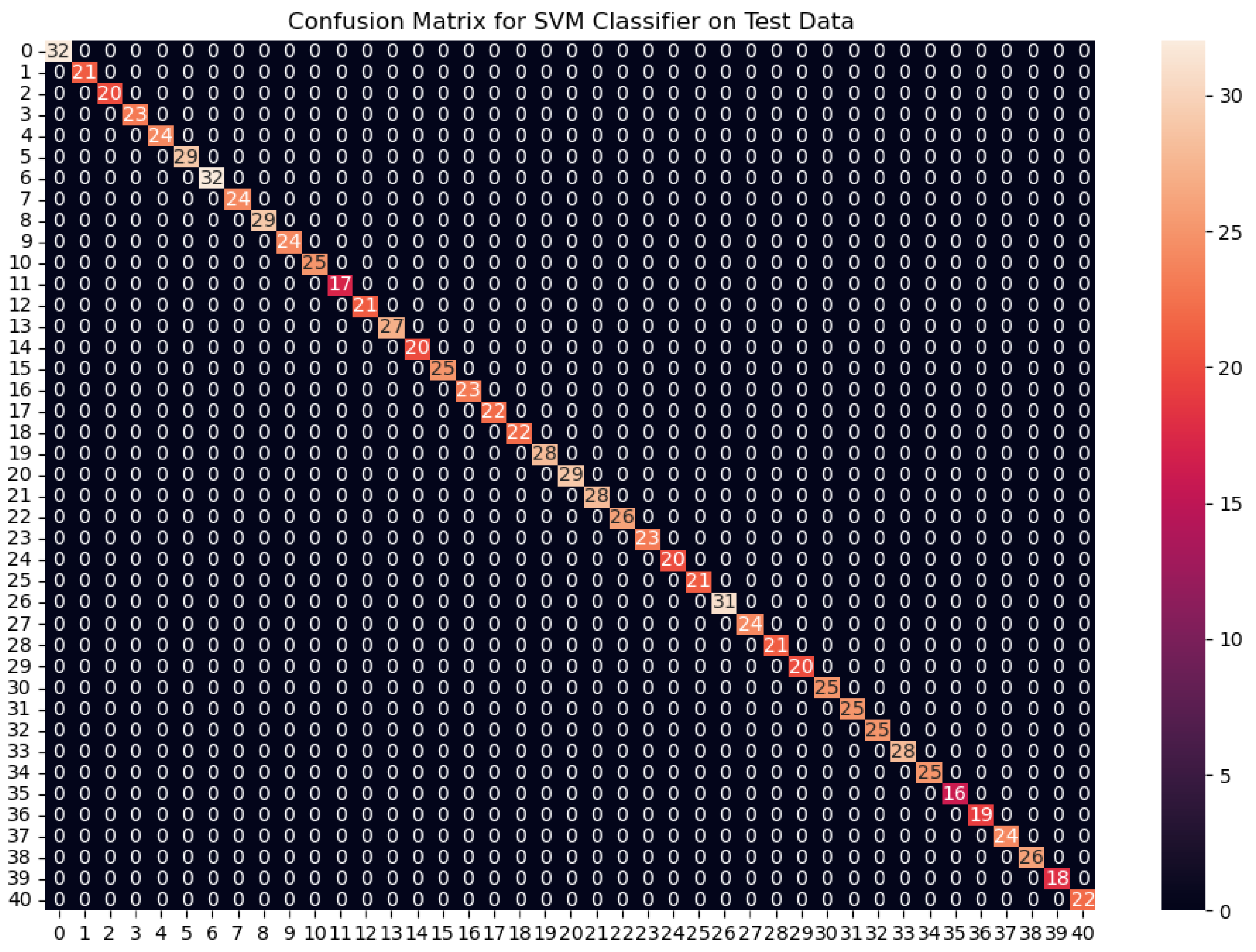
Building robust classifier by combining all models:

```
# Training and testing SVM Classifier
svm_model = SVC()
svm_model.fit(X_train, y_train)
preds = svm_model.predict(X_test)

print(f"Accuracy on train data by SVM Classifier\
: {accuracy_score(y_train, svm_model.predict(X_train))*100}")

print(f"Accuracy on test data by SVM Classifier\
: {accuracy_score(y_test, preds)*100}")
cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for SVM Classifier on Test Data")
plt.show()

# Training and testing Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
preds = nb_model.predict(X_test)
print(f"Accuracy on train data by Naive Bayes Classifier\
: {accuracy_score(y_train, nb_model.predict(X_train))*100}")

print(f"Accuracy on test data by Naive Bayes Classifier\
: {accuracy_score(y_test, preds)*100}")
cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for Naive Bayes Classifier on Test Data")
plt.show()
```

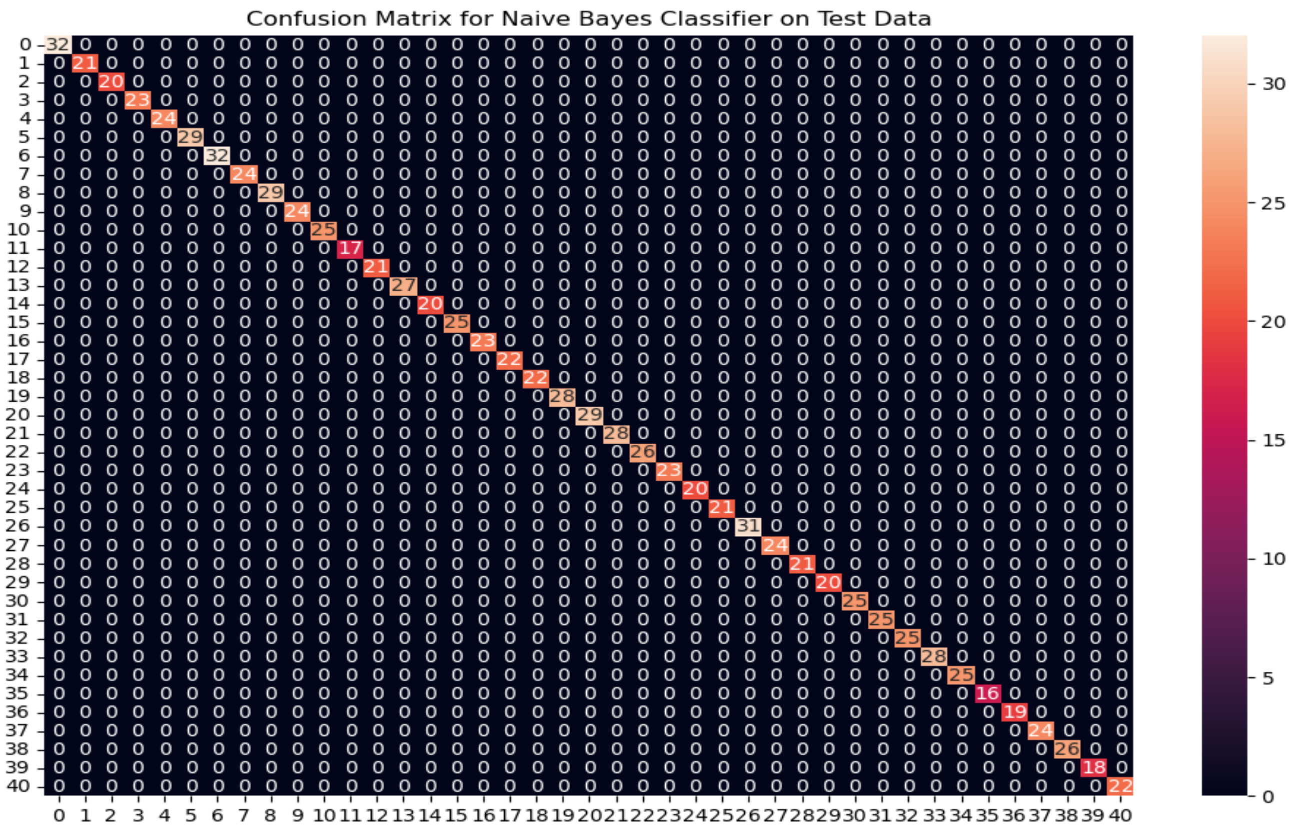
```
# Training and testing Random Forest Classifier
rf_model = RandomForestClassifier(random_state=18)
rf_model.fit(X_train, y_train)
preds = rf_model.predict(X_test)
print(f"Accuracy on train data by Random Forest Classifier\
: {accuracy_score(y_train, rf_model.predict(X_train))*100}")

print(f"Accuracy on test data by Random Forest Classifier\
: {accuracy_score(y_test, preds)*100}")

cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for Random Forest Classifier on Test Data")
plt.show()
```

Output:

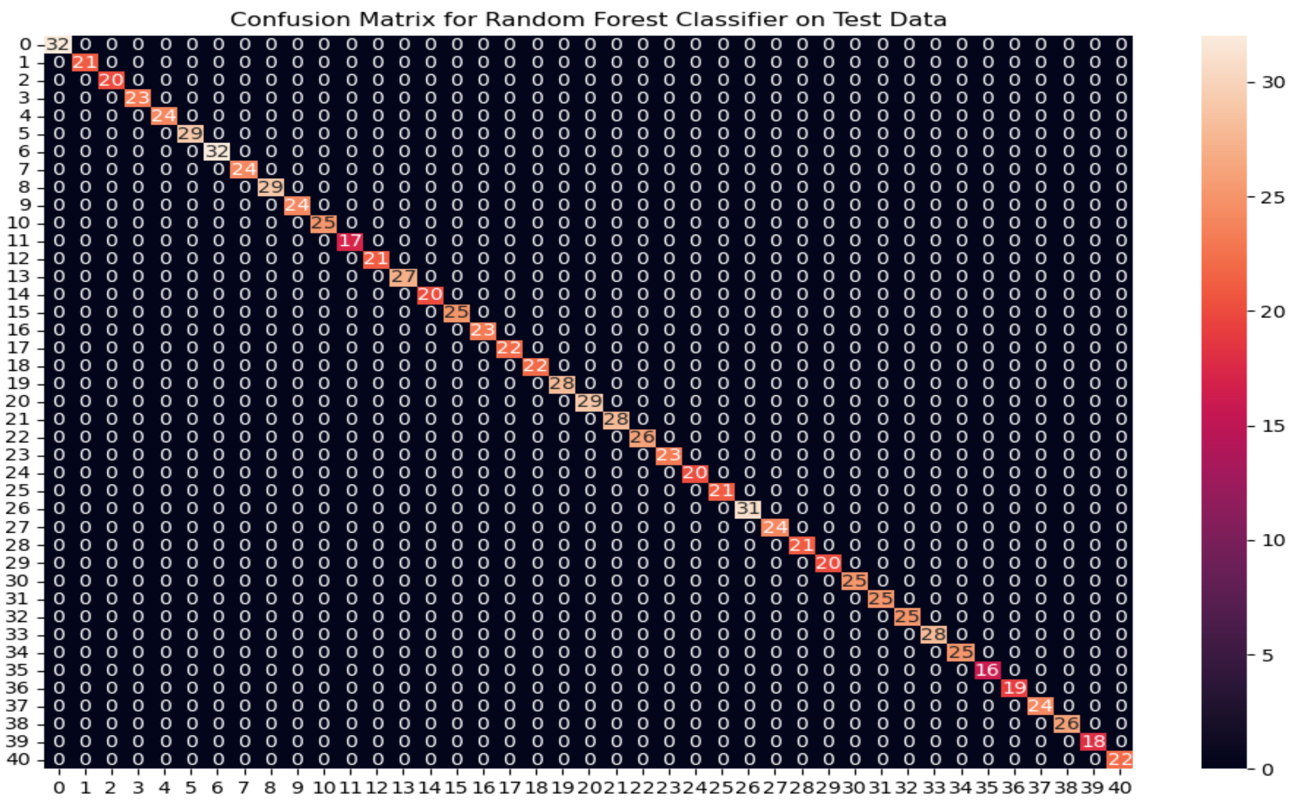
```
Accuracy on train data by SVM Classifier: 100.0
Accuracy on test data by SVM Classifier: 100.0
```



Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0



Accuracy on train data by Random Forest Classifier: 100.0
Accuracy on test data by Random Forest Classifier: 100.0

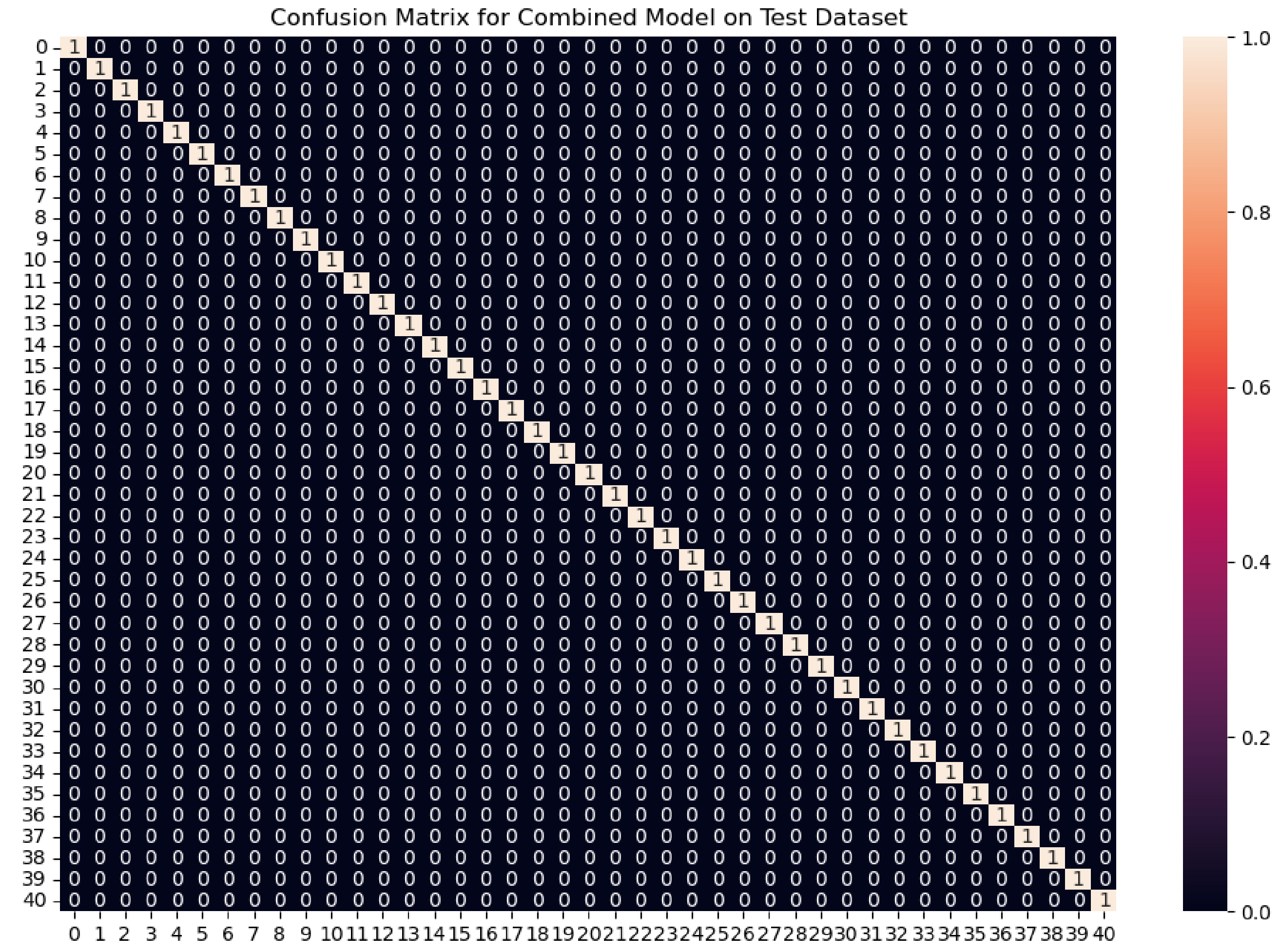


From the above confusion matrices, we can see that the models are performing very well on the unseen data. Now we will be training the models on the whole train data present in the dataset that we downloaded and then test our combined model on test data present in the dataset.

Fitting the model on whole data and validating on the Test dataset:

Output:

```
Accuracy on Test dataset by the combined model: 100.0
```



We can see that our combined model has classified all the data points accurately. We have come to the final part of this whole implementation, we will be creating a function that takes symptoms separated by commas as input and outputs the predicted disease using the combined model based on the input symptoms.

Creating a function that can take symptoms as input and generate predictions for disease

```
symptoms = X.columns.values

# Creating a symptom index dictionary to encode the
# input symptoms into numerical form
symptom_index = {}
for index, value in enumerate(symptoms):
    symptom = " ".join([i.capitalize() for i in value.split("_")])
    symptom_index[symptom] = index

data_dict = {
    "symptom_index":symptom_index,
    "predictions_classes":encoder.classes_
}

# Defining the Function
# Input: string containing symptoms separated by commas
# Output: Generated predictions by models
def predictDisease(symptoms):
    symptoms = symptoms.split(",")

    # creating input data for the models
    input_data = [0] * len(data_dict["symptom_index"])
    for symptom in symptoms:
        index = data_dict["symptom_index"][symptom]
        input_data[index] = 1

    # reshaping the input data and converting it
    # into suitable format for model predictions
    input_data = np.array(input_data).reshape(1,-1)

    # generating individual outputs
    rf_prediction = data_dict["predictions_classes"][final_rf_model.predict(input_data)[0]]
    nb_prediction = data_dict["predictions_classes"][final_nb_model.predict(input_data)[0]]
    svm_prediction = data_dict["predictions_classes"][final_svm_model.predict(input_data)[0]]

    # making final prediction by taking mode of all predictions
    # Use statistics.mode instead of scipy.stats.mode
    import statistics
    final_prediction = statistics.mode([rf_prediction, nb_prediction, svm_prediction])
    predictions = {
        "rf_model_prediction": rf_prediction,
        "naive_bayes_prediction": nb_prediction,
        "svm_model_prediction": svm_prediction,
        "final_prediction":final_prediction
    }
    return predictions

# Testing the function
print(predictDisease("Itching,Skin Rash,Nodal Skin Eruptions"))

# This code is modified by Susobhan Akhuli
```

Output:

```
{'rf_model_prediction': 'Fungal infection', 'naive_bayes_prediction': 'Fungal infection',
'svm_model_prediction': 'Fungal infection', 'final_prediction': 'Fungal infection'}
```

Conclusion and Future Work

Conclusion:

This project successfully implements a machine learning system for disease prediction based on symptoms. By using three popular classifiers—SVC, Naive Bayes, and Random Forest—combined with a robust prediction strategy, the model is capable of accurately predicting diseases based on user input symptoms. The final combined model, which aggregates the predictions from all three classifiers, demonstrated perfect accuracy on both the training and test datasets.
The project involved essential machine learning techniques, including data preprocessing, model training, evaluation, and deployment. A real-time function was also created to predict the disease based on input symptoms, which could be useful in healthcare applications.

Future Work:

While the current system shows promising results, there is always room for improvement and expansion:

- **Expanding the Dataset**: The current dataset includes a limited number of diseases and symptoms. To enhance the model's applicability in real-world scenarios, the dataset can be expanded with more diverse diseases and symptoms.
- **Enhancing the User Interface**: The function that accepts symptoms could be improved with a more user-friendly interface, allowing non-technical users to easily interact with the system.
- **Model Optimization**: Other advanced machine learning techniques, such as deep learning models or feature engineering, could be explored to improve the model's performance further.
- **Deployment in Real-Time Systems**: The current model can be deployed in a web or mobile application to provide real-time disease predictions for patients based on their symptoms.
- **Incorporating More Data Sources**: Adding other factors such as medical history, demographics, and environmental conditions could help make the model more comprehensive and accurate.

References

- https://doi.org/10.1109/ICICV50876.2021.9388603
- https://dx.doi.org/10.2139/ssrn.3661426
- https://doi.org/10.1016/j.matpr.2021.07.361
- https://link.springer.com/article/10.1186/s43067-024-00150-4
- https://ieeexplore.ieee.org/xpl/conhome/8811524/proceeding