CAP 5768 INTRO TO DATA SCIENCE

DISEASE PREDICTION USING MACHINE LEARNING

Name: Sai Supraja Chinthapalli

ZNumber: Z23760554

Project Overview

Because of the complexity of symptoms and patient histories, healthcare systems have difficulty making early disease diagnoses. Misdiagnosis or delayed diagnosis can lead to ineffective treatments, higher medical costs, and unfavorable patient outcomes. This project aims to develop a disease prediction system based on machine learning that can assist physicians in predicting potential illnesses based on patient symptoms, test results, and medical history. A tool like this can improve clinical decision-making, cut down on diagnostic errors, and speed up patient care.

1.Data Collection

The goal is to develop a predictive model that classifies potential diseases using a combination of symptoms reported by patients. It is difficult to tell many diseases apart accurately because many of their symptoms overlap. In situations where access to skilled medical professionals is restricted, a dependable predictive system can act as a diagnostic aid.

Source: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning/data

Dataset Description:

We employ a Kaggle-sourced publicly accessible dataset to address this issue. The dataset consists of two main components:

Training.csv contains 4,920 patient records with 132 binary features, each of which indicates whether a particular symptom is present (one) or absent (zero). The diagnosis of each patient's disease is displayed in the final column, prognosis.

Testing.csv includes a smaller set of patient records with the same format, used for model validation and performance assessment on unseen data.

The medical history of each individual is depicted in the dataset's rows. Variables include itching, joint pain, fatigue, vomiting, and other symptoms. Numerous diseases, including

fungi, allergies, diabetes, migraine, and tuberculosis, make up the target variable (prognosis). This structured dataset is well-suited for supervised classification algorithms and enables the development of an end-to-end disease prediction pipeline.

2.Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential first step in the data science workflow because it sheds light on the structure of the dataset, its feature distributions, missing values, and potential connections between the target variable and predictors.

2.1 Dataset Dimensions And Structure

There are 134 columns and 4,920 rows in the training dataset. 132 of these are binary symptom indicators, with 1 denoting a symptom's presence and 0 denoting its absence. The disease diagnosis is the target variable in the prognosis column. An additional column, Unnamed: 133, appears to be empty and will be dropped during preprocessing.

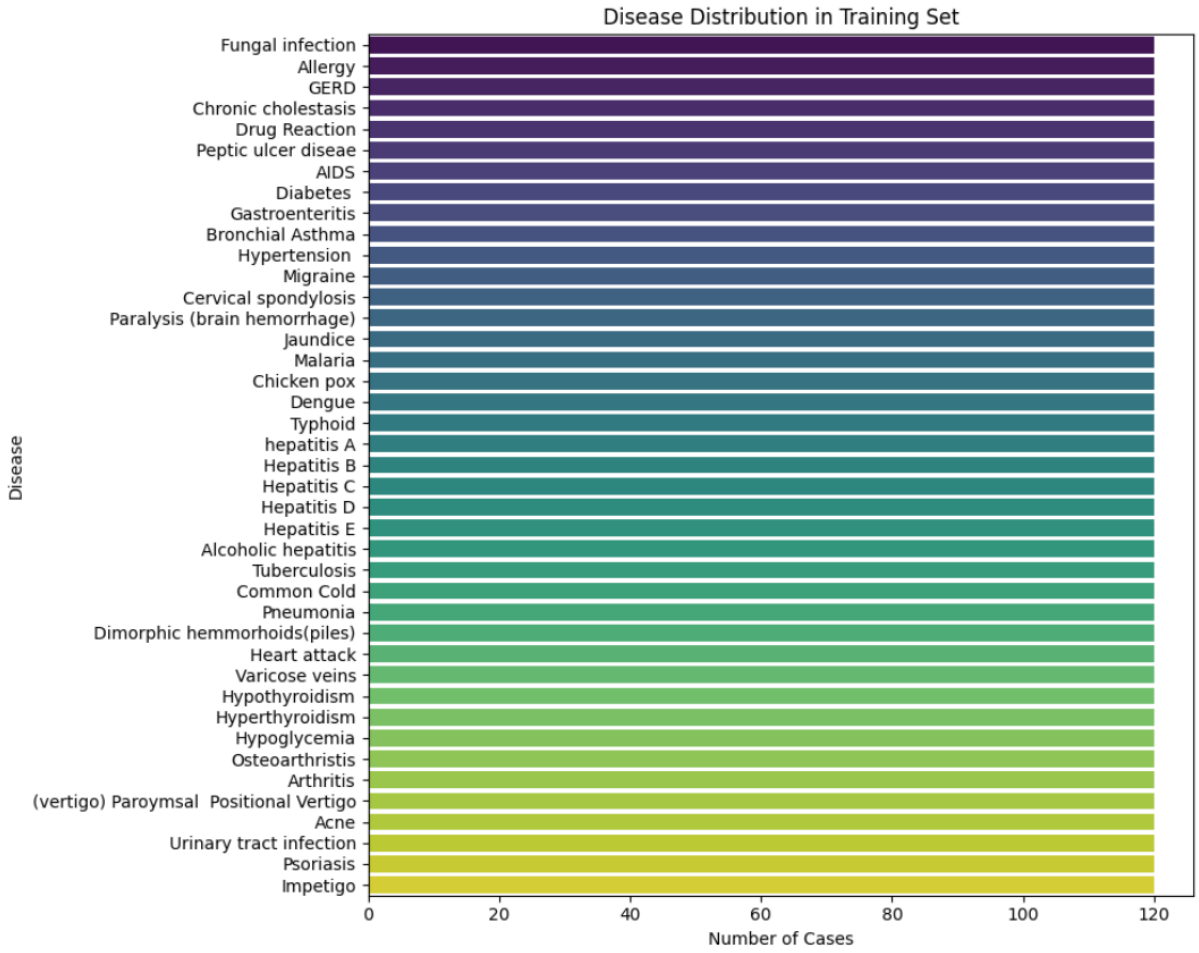
2.2 Target Variable Distribution

The dataset contains multiple disease classes, with prognosis as the categorical target variable. Below is a visualization of the number of instances per disease:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming our data is in a DataFrame named 'df' and the column with diseases is named 'prognosis'
# Replace 'df' and 'prognosis' with actual DataFrame and column name
# Create a Pandas Series with the disease counts

# Load dataframe here.

df = pd.read_csv('/content/Training.csv')

prognosis_counts = df['prognosis'].value_counts()

plt.figure(figsize=(10, 8))
sns.barplot(y=prognosis_counts.index, x=prognosis_counts.values, palette="viridis")
plt.title("Disease Distribution in Training Set")
plt.xlabel("Number of Cases")
plt.ylabel("Disease")
plt.tight_layout()
plt.show()
```

This step helps identify any class imbalance that may affect model training and evaluation.

Output:



2.3 Missing Values

The dataset is largely clean, with the exception of the final column Unnamed: 133, which contains only null values and will be removed. No other columns show missing data.

2.4 Symptom Frequency Analysis

We compute the sum of each symptom in all patient records to comprehend symptom prevalence. The most typical symptoms are:

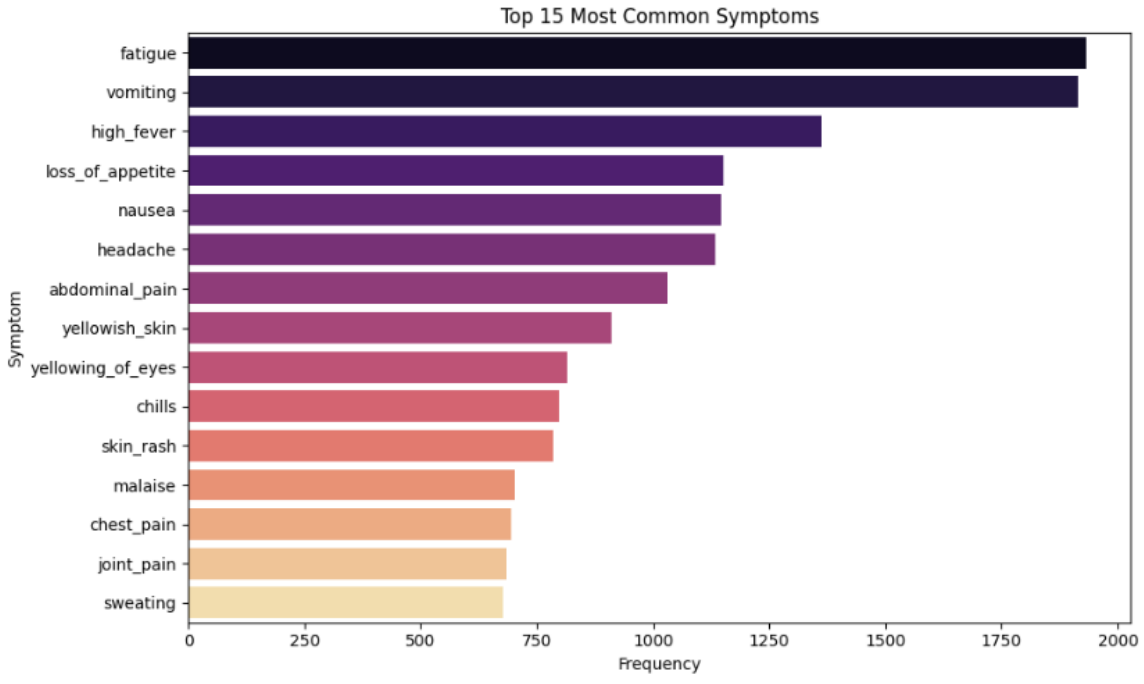
- Fatigue
- Cough
- High fever
- Headache

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming symptom columns start from the second column (index 1)

# Load the dataframe
df = pd.read_csv('/content/Training.csv')

# Convert columns to numeric, errors='coerce' will replace non-numeric values with N
numeric_df = df.iloc[:, 1:].apply(pd.to_numeric, errors='coerce')

symptom_sums = numeric_df.sum().sort_values(ascending=False)

# Continue with plotting code:
top_symptoms = symptom_sums.head(15)
plt.figure(figsize=(10, 6))
sns.barplot(x=top_symptoms.values, y=top_symptoms.index, palette="magma")
plt.title("Top 15 Most Common Symptoms")
plt.xlabel("Frequency")
plt.ylabel("Symptom")
plt.tight_layout()
plt.show()
```

This may decrease the diagnostic value of symptoms that are highly correlated with multiple conditions.

Below is a visualization of the top 15 most frequent symptoms:

Output:



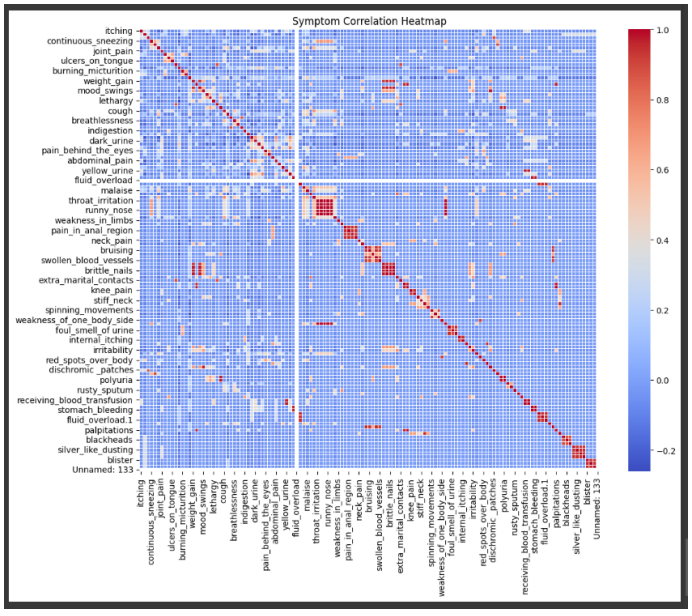
2.5 Feature Correlation

Correlation heatmaps or pairwise analysis can still be used to identify symptom co-occurrence even though all symptom features are binary. For instance, vomiting and nausea might frequently appear together.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

df = pd.read_csv('/content/Training.csv')

# Select only numeric columns for correlation calculation
numeric_df = df.select_dtypes(include=np.number)

# Calculate the correlation matrix using the numeric DataFrame
correlation_matrix = numeric_df.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, cmap='coolwarm', linewidths=0.5)
plt.title('Symptom Correlation Heatmap')
plt.tight_layout()
plt.show()
```

This heatmap helps detect which symptoms tend to co-occur, which can provide valuable insight for feature selection or dimensionality reduction later in the pipeline.

Output:



3. Data Preprocessing

Data preprocessing is a crucial step to prepare raw data for model training and ensure that it is clean, consistent, and suitable for analysis.

3.1 Handling Missing Data

The column Unnamed: 133, which contained only missing values, was dropped from the dataset as it provided no useful information.

Missing values were found in all other features. Since the features are binary indicators of symptoms (1 = present, 0 = absent), missing values - if found - were logically assumed to represent the absence of a symptom and were therefore filled with 0.

```
import pandas as pd

# Load the dataframe
df = pd.read_csv('/content/Training.csv')

# Drop column with only missing values
df.drop(columns=['Unnamed: 133'], inplace=True)

# Impute missing binary symptom values with 0
df.fillna(0, inplace=True)
```

3.2 Encoding Categorical Target Variable

The disease names are formatted as strings in the target column prognosis. Label Encoding was used to turn these into numerical labels, making them compatible with scikit-learn machine learning models.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load the dataframe
df = pd.read_csv('/content/Training.csv')

# use 'df'
le = LabelEncoder()
df['prognosis'] = le.fit_transform(df['prognosis'])
```

3.3 Feature And Target Separation

We separated the target variable from the input features in preparation for training:

```
X = df.drop(['prognosis'], axis=1)  # Features
y = df['prognosis']                 # Target
```

3.4 Outlier Detection And Treatment

Traditional methods for identifying outliers are inapplicable because all features are binary (0 or 1) in this case. However, outliers would be identified using methods like the following if subsequent versions of the dataset included continuous numerical features like age or test results: Z-score Technique Method with Interquartile Range (IQR)

3.5 Scaling And Normalization

Scaling and normalization are typically utilized to standardize numerical features in algorithms sensitive to feature magnitudes. Scaling is not required for binary features in this scenario. However, any additional continuous features in the future can be processed for completeness. Standard Normalization with the StandardScaler. For min-max scaling, MinMaxScaler

3.6 Splitting The Dataset

The dataset was divided into training and testing subsets to prevent overfitting and ensure a fair evaluation:

```
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=42)
```

As a result, we get: 80 percent of the data used to train the model, 20% is set aside for testing and validating models.

4. Feature Engineering

By reducing dimensionality and creating more informative variables, feature engineering improves model performance.

4.1 Creating New Features

We added a new feature called symptom_count, which shows how many symptoms each patient has.

```
X_train['symptom_count'] = X_train.sum(axis=1)
X_test['symptom_count'] = X_test.sum(axis=1)
```

4.2 Dimensionality Reduction With PCA

Principal Component Analysis (PCA) helps reduce feature space while maintaining variance because we have more than 130 symptom features.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer

# Load the dataframe
df = pd.read_csv('/content/Training.csv')

# use 'df'
le = LabelEncoder()
df['prognosis'] = le.fit_transform(df['prognosis'])

X = df.drop(['prognosis'], axis=1)  # Features
y = df['prognosis']                 # Target

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

# Impute missing values using SimpleImputer before applying PCA
imputer = SimpleImputer(strategy='mean')  # or strategy='most_frequent' for categorical data
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Get the column names from X_train after imputation
X_train_columns = [f"column_{i}" for i in range(X_train.shape[1])]  # Generate column names

# Convert the NumPy arrays back to DataFrames
X_train = pd.DataFrame(X_train, columns=X_train_columns)
X_test = pd.DataFrame(X_test, columns=X_train_columns)

X_train['symptom_count'] = X_train.sum(axis=1)
X_test['symptom_count'] = X_test.sum(axis=1)

pca = PCA(n_components=0.95)  # retain 95% variance
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Print the outputs
print("X_train_pca:")
print(X_train_pca)
print("\nX_test_pca:")
print(X_test_pca)
```

Output

```
X_train_pca:
[[ 2.68594409e+00 -5.82807316e-01  3.91807641e-01 ... -2.01332422e-01
   2.55077717e-01  2.01703362e-01]
 [ 2.67032691e+00  4.08684653e-01  5.79442382e-01 ...  6.99766988e-01
   2.99069821e-01 -2.48030013e-02]
 [ 3.65614108e+00  9.74939162e-01 -6.14260057e-01 ... -3.97174191e-01
  -1.86044935e-01 -1.13703119e-01]
 ...
 [ 6.73462297e+00  3.28661514e-01  9.70574188e-01 ... -5.07264410e-01
  -2.49572355e-01 -2.41079835e-01]
 [-3.55318869e+00 -1.91587173e-01  3.56768786e-02 ...  4.65560540e-01
  -3.52202898e-01  1.15417108e-01]
 [-2.58556261e+00 -2.43447065e-01 -4.03246341e-03 ...  2.53767723e-01
   2.42441216e-01  3.91916879e-02]]

X_test_pca:
[[-4.53075946 -0.2802825   0.10691161 ...  0.27569217 -0.26051765
  -0.05991708]
 [-3.55864105 -0.31420265  0.05461802 ...  0.37759031 -0.32661197
  -0.08331339]
 [ 3.57311424 -1.05425183 -1.64461185 ... -0.21510342  0.058947
  -0.0429544 ]
 ...
 [ 6.73462297  0.32866151  0.97057419 ... -0.50726441 -0.24957236
  -0.24107984]
 [-4.53067355 -0.28018412  0.10685964 ...  0.27488308 -0.25972901
  -0.05972541]
 [-4.49613806  0.10083338  0.19920766 ... -0.3729493  -0.17993103
   0.16757239]]
```

4.3 Feature Selection Using Correlation And RFE

We eliminated redundant (highly correlated) features by employing correlation analysis. Using a basic model, the most predictive features were chosen using Recursive Feature Elimination (RFE).

```python
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
rfe = RFE(model, n_features_to_select=50)
rfe.fit(X_train, y_train)
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)
print("X_train_rfe:")
print(X_train_rfe)
print("\nX_test_rfe:")
print(X_test_rfe)
```

Output

```
X_train_rfe:
[[ 1.  1.  0. ...  0.  0. 10.]
 [ 0.  0.  0. ...  0.  0. 10.]
 [ 1.  0.  0. ...  0.  0. 11.]
 ...
 [ 0.  1.  0. ...  0.  0. 14.]
 [ 1.  1.  0. ...  0.  0.  4.]
 [ 1.  1.  0. ...  0.  0.  5.]]

X_test_rfe:
[[ 0.  1.  0. ...  0.  0.  3.]
 [ 0.  1.  0. ...  0.  0.  4.]
 [ 0.  0.  0. ...  0.  0. 11.]
 ...
 [ 0.  1.  0. ...  0.  0. 14.]
 [ 0.  1.  0. ...  0.  0.  3.]
 [ 0.  0.  0. ...  0.  0.  3.]]
```

By taking these steps, we make sure that we keep only the most important and non-redundant features, making the model more efficient and easier to understand.

5. Model Building

Using Scikit-learn, we implemented both classification and clustering algorithms to assess the model's performance.

5.1 Classification Models

Random Forest Classifier

The Random Forest algorithm is an ensemble method that builds multiple decision trees during training and outputs the class that is the mode of the classes output by individual trees. It provides high accuracy, handles large datasets with higher dimensionality, and prevents overfitting.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_rfe, y_train)
y_pred_rf = rf.predict(X_test_rfe)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

Output

```
Random Forest Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        25
           4       1.00      1.00      1.00        24
           5       1.00      1.00      1.00        23
           6       1.00      1.00      1.00        33
           7       1.00      1.00      1.00        23
           8       1.00      1.00      1.00        21
           9       1.00      1.00      1.00        15
          10       1.00      1.00      1.00        23
          11       1.00      1.00      1.00        26
          12       1.00      1.00      1.00        21
          13       1.00      1.00      1.00        29
          14       1.00      1.00      1.00        24
          15       1.00      1.00      1.00        19
          16       1.00      1.00      1.00        28
          17       1.00      1.00      1.00        25
          18       1.00      1.00      1.00        23
          19       1.00      1.00      1.00        27
          20       1.00      1.00      1.00        26
          21       1.00      1.00      1.00        23
          22       1.00      1.00      1.00        29
          23       1.00      1.00      1.00        25
          24       1.00      1.00      1.00        24
          25       1.00      1.00      1.00        26
          26       1.00      1.00      1.00        21
          27       1.00      1.00      1.00        24
          28       1.00      1.00      1.00        19
          29       1.00      1.00      1.00        22
          30       1.00      1.00      1.00        25
          31       1.00      1.00      1.00        22
          32       1.00      1.00      1.00        24
          33       1.00      1.00      1.00        17
          34       1.00      1.00      1.00        28
          35       1.00      1.00      1.00        22
          36       1.00      1.00      1.00        25
          37       1.00      1.00      1.00        19
          38       1.00      1.00      1.00        26
          39       1.00      1.00      1.00        22
          40       1.00      1.00      1.00        34

    accuracy                           1.00       984
   macro avg       1.00      1.00      1.00       984
weighted avg       1.00      1.00      1.00       984
```

Support Vector Machine (SVM)

A supervised learning algorithm known as SVM seeks the optimal hyperplane for class separation in the feature space. Through the kernel trick, it works best in high-dimensional datasets and situations where the decision boundary is not linearly separable.

```python
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train_rfe, y_train)
y_pred_svm = svm.predict(X_test_rfe)

print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

Output

SVM Accuracy: 0.9939024390243902

	precision	recall	f1-score	support
0	1.00	1.00	1.00	18
1	1.00	1.00	1.00	30
2	1.00	1.00	1.00	24
3	1.00	1.00	1.00	25
4	1.00	1.00	1.00	24
5	1.00	1.00	1.00	23
6	1.00	1.00	1.00	33
7	1.00	1.00	1.00	23
8	1.00	1.00	1.00	21
9	1.00	1.00	1.00	15
10	1.00	1.00	1.00	23
11	1.00	1.00	1.00	26
12	1.00	1.00	1.00	21
13	1.00	0.90	0.95	29
14	1.00	0.88	0.93	24
15	0.86	1.00	0.93	19
16	1.00	1.00	1.00	28
17	1.00	1.00	1.00	25
18	1.00	1.00	1.00	23
19	1.00	1.00	1.00	27
20	1.00	1.00	1.00	26
21	1.00	1.00	1.00	23
22	1.00	1.00	1.00	29
23	1.00	1.00	1.00	25
24	1.00	1.00	1.00	24
25	1.00	1.00	1.00	26
26	1.00	1.00	1.00	21
27	1.00	1.00	1.00	24
28	1.00	1.00	1.00	19
29	1.00	1.00	1.00	22
30	1.00	1.00	1.00	25
31	1.00	1.00	1.00	22
32	1.00	1.00	1.00	24
33	1.00	1.00	1.00	17
34	1.00	1.00	1.00	28
35	1.00	1.00	1.00	22
36	1.00	1.00	1.00	25
37	1.00	1.00	1.00	19
38	0.90	1.00	0.95	26
39	1.00	1.00	1.00	22
40	1.00	1.00	1.00	34
accuracy			0.99	984
macro avg	0.99	0.99	0.99	984
weighted avg	0.99	0.99	0.99	984

5.2 Clustering Model

K-Means

An unsupervised algorithm known as K-Means divides the dataset into K clusters by reducing variance within each cluster. Even without labeled data, it is useful for grouping similar symptom patterns.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(X_train_rfe)
labels = kmeans.labels_

score = silhouette_score(X_train_rfe, labels)
print("Silhouette Score for KMeans:", score)
```

Output

```
Silhouette Score for KMeans: 0.2408493745497029
```

5.3 Regression Models

Linear Regression

By fitting a linear equation, linear regression models the relationship between a dependent variable and one or more independent variables. Even though our task is classification, regression can be used to predict the likelihood of disease or the severity of symptoms.

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train_rfe, y_train)
y_pred_lr = lr.predict(X_test_rfe)

print("Linear Regression Predictions:")
print(y_pred_lr)
```

Output

```
Linear Regression Predictions:
[ 2.72401404  1.74401661 25.22292501  0.75416313 11.67166793 22.968595
 25.05348031  3.48166257 18.34697363 29.85595475 37.73573103 12.33823087
 26.35856002 13.2343955   3.84888202  2.86888458 14.72366792 22.45862412
 13.31822831  6.83608815 10.49577283 33.72559718 18.34697363 10.43260694
  4.53120774 14.67055908 29.03905546 30.83595219 26.57763286 29.85595475
 11.67166793 18.34697363 19.61674396 23.57829016 39.77197659 28.57767036
 26.48997428 13.85498209 29.03905546 37.73573103 11.95404038 -0.30203023
 26.06766426 11.05994648 24.57048745 18.63674652 18.41410083 18.34697363
 34.90910994 34.90910994 19.82678626 35.36777636 38.79197915  5.51120518
 25.22292501 15.57130764 35.57424274 13.02591794 37.73573103  9.07831171
 25.05348031 26.20292245 29.66397229 19.82678626  0.67796721 26.57763286
 31.31941683  1.73416057  8.69809656 15.70366535  9.75942617 10.43260694
  1.74401661 23.43862156 26.57763286 14.57632498 26.66987243 26.66987243
 16.1249872  29.85595475 18.9451771  26.66987243 16.44476121 12.87498465
 18.34697363  0.75416313 24.57048745 18.63674652  2.19838566  8.69927208
 25.59763542 31.5099612  29.93932944  2.86888458 37.28726319 18.63674652
 31.5099612  19.82678626 26.57883652 12.34924527 22.968595   10.43260694
  2.86888458  4.53120774 29.16268415 36.87249232 16.44476121 29.85595475
 22.45862412  0.75416313 29.93932944 10.77723738 19.82678626 25.05348031
 12.33823087 12.87498465 26.57763286 23.12822011 23.13399348  3.84888202
  6.83608815 34.90910994 31.31941683 10.18498593  4.55700739 29.03905546
 19.61674396 26.20292245 32.37880278 31.11221833 11.34323633  4.14314256
  6.83608815 12.39326261 12.65166537  1.74401661 18.34697363 29.03905546
 14.72366792 36.55424018 32.52443833 19.82678626 34.90910994 18.34697363
 22.99680585 31.11221833 12.87498465 39.77197659 37.28726319  3.84888202
 13.31822831 31.11221833 32.52443833 31.95697548 26.06766426 38.07465864
 25.05348031 26.35856002  2.86888458 19.16954618  8.51312011 19.15753323
  1.73416057 19.82678626 37.08735135 35.57424274 34.60903046 10.43260694
 26.66987243 23.94464062 27.7479318  26.35856002 29.7549103  18.63674652
 22.39967597 26.66987243  1.87382917 37.08735135 13.02591794 13.76575536
 31.5099612   0.75416313 19.16954618 37.08735135 29.85595475 32.21248857
 12.33823087 18.34697363  3.85777592 38.71572847 35.57424274  1.85767369
 39.77197659 29.03905546 22.45862412 10.18498593  9.75942617  6.83608815
 21.25163067 31.11221833 18.63674652 37.08735135 36.87249232 11.22957924
 14.59573787  2.79703903 16.44476121  4.53120774 11.67166793 10.18498593
 18.34697363 32.09221577 19.82678626 14.72366792 18.63674652 23.12822011
 31.5099612  22.32720215  8.30991376 26.35856002 12.34924527  2.86888458
  3.84888202 22.968595   31.31941683 29.7549103  10.18498593 21.13716241
 25.59763542 34.90910994  4.55700739 14.72366792 35.57424274 30.0190529
 13.2343955  12.87498465 31.11221833 10.43260694 13.31822831  3.48166257
 16.1249872  15.57130764 -2.84059558 38.79197915 32.29941427 39.78767909
```

Random Forest Regressor

The Random Forest Regressor is used to predict continuous outputs in a manner that is comparable to that of its classification counterpart. For increased robustness, it combines predictions from multiple decision trees.

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train_rfe, y_train)
y_pred_rfr = rfr.predict(X_test_rfe)

print("Random Forest Regressor:")
print(y_pred_lr)
```

Output

```
Random Forest Regressor:
[ 2.72401404  1.74401661 25.22292501  0.75416313 11.67166793 22.968595
 25.05348031  3.48166257 18.34697363 29.85595475 37.73573103 12.33823087
 26.35856002 13.2343955   3.84888202  2.86888458 14.72366792 22.45862412
 13.31822831  6.83608815 10.49577283 33.72559718 18.34697363 10.43260694
  4.53120774 14.67055908 29.03905546 30.83595219 26.57763286 29.85595475
 11.67166793 18.34697363 19.61674396 23.57829016 39.77197659 28.57767036
 26.48997428 13.85498209 29.03905546 37.73573103 11.95404038 -0.30203023
 26.06766426 11.05994648 24.57048745 18.63674652 18.41410083 18.34697363
 34.90910994 34.90910994 19.82678626 35.36777636 38.79197915  5.51120518
 25.22292501 15.57130764 35.57424274 13.02591794 37.73573103  9.07831171
 25.05348031 26.20292245 29.66397229 19.82678626  0.67796721 26.57763286
 31.31941683  1.73416057  8.69809656 15.70366535  9.75942617 10.43260694
  1.74401661 23.43862156 26.57763286 14.57632498 26.66987243 26.66987243
 16.1249872  29.85595475 18.9451771  26.66987243 16.44476121 12.87498465
 18.34697363  0.75416313 24.57048745 18.63674652  2.19838566  8.69927208
 25.59763542 31.5099612  29.93932944  2.86888458 37.28726319 18.63674652
 31.5099612  19.82678626 26.57883652 12.34924527 22.968595   10.43260694
  2.86888458  4.53120774 29.16268415 36.87249232 16.44476121 29.85595475
 22.45862412  0.75416313 29.93932944 10.77723738 19.82678626 25.05348031
 12.33823087 12.87498465 26.57763286 23.12822011 23.13399348  3.84888202
  6.83608815 34.90910994 31.31941683 10.18498593  4.55700739 29.03905546
 19.61674396 26.20292245 32.37880278 31.11221833 11.34323633  4.14314256
  6.83608815 12.39326261 12.65166537  1.74401661 18.34697363 29.03905546
 14.72366792 36.55424018 32.52443833 19.82678626 34.90910994 18.34697363
 22.99680585 31.11221833 12.87498465 39.77197659 37.28726319  3.84888202
 13.31822831 31.11221833 32.52443833 31.95697548 26.06766426 38.07465864
 25.05348031 26.35856002  2.86888458 19.16954618  8.51312011 19.15753323
  1.73416057 19.82678626 37.08735135 35.57424274 34.60903046 10.43260694
 26.66987243 23.94464062 27.7479318  26.35856002 29.7549103  18.63674652
 22.39967597 26.66987243  1.87382917 37.08735135 13.02591794 13.76575536
 31.5099612   0.75416313 19.16954618 37.08735135 29.85595475 32.21248857
 12.33823087 18.34697363  3.85777592 38.71572847 35.57424274  1.85767369
 39.77197659 29.03905546 22.45862412 10.18498593  9.75942617  6.83608815
 21.25163067 31.11221833 18.63674652 37.08735135 36.87249232 11.22957924
 14.59573787  2.79703903 16.44476121  4.53120774 11.67166793 10.18498593
 18.34697363 32.09221577 19.82678626 14.72366792 18.63674652 23.12822011
 31.5099612  22.32720215  8.30991376 26.35856002 12.34924527  2.86888458
  3.84888202 22.968595   31.31941683 29.7549103  10.18498593 21.13716241
 25.59763542 34.90910994  4.55700739 14.72366792 35.57424274 30.0190529
 13.2343955  12.87498465 31.11221833 10.43260694 13.31822831  3.48166257
```

6. Model Evaluation And Hyperparameter Tuning

To select the best machine learning model, it is essential to evaluate their performance after building multiple models. The two most important tasks in this step are determining how well the models adapt to new data and optimizing their parameters for optimal performance.

6.1 Cross-Validation and Hyperparameter Tuning

We employ K-Fold Cross-Validation, which divides the training dataset into k subsets, to avoid overfitting and guarantee model robustness. Through each fold, the model is trained on k–1 subsets and validated on the remaining one. After repeating this procedure k times, the average performance metric is determined.

This is combined with Grid Search, a method that tests every possible combination of specified hyperparameters to determine which one performs best in validation.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the hyperparameter grid
param_grid = {'n_estimators': [50, 100, 150], 'max_depth': [10, 20, None]}

# Create a GridSearchCV object
gs = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5)

# Fit the model
gs.fit(X_train_rfe, y_train)

# Get the best hyperparameters and score
best_params = gs.best_params_
best_score = gs.best_score_

# Print the results
print("Best Hyperparameters:", best_params)
print("Best Cross-Validation Score:", best_score)

# Get the best model
best_rf = gs.best_estimator_
```

Output

```
Best Hyperparameters: {'max_depth': None, 'n_estimators': 50}
Best Cross-Validation Score: 1.0
```

Here the number of trees in the forest is determined by n_estimators. Maximum depth of each tree is controlled by max_depth. The most effective model is saved as best_rf.

6.2 Evaluation Metrics

We employ a number of common metrics to evaluate classification models.

Confusion Matrix

It categorizes the number of correct and incorrect predictions based on the labels of the actual and predicted events. Because of this, we can compute:

Accuracy: (TP + TN) / Total

Precision: TP / (TP + FP)

Recall: TP / (TP + FN)

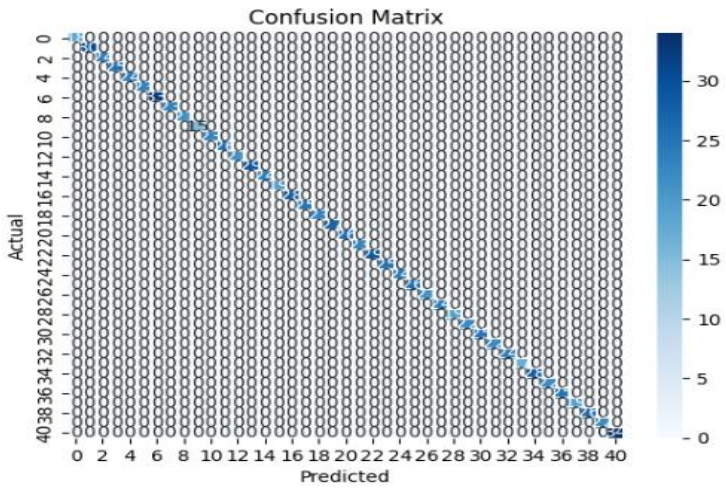
The F1-Score is the harmonic mean of recall and precision.

Classification Report

Summarizes precision, recall, F1-score, and support (number of actual occurrences) for each class.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve

# Confusion Matrix
cm = confusion_matrix(y_test, best_rf.predict(X_test_rfe))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Classification Report
print(classification_report(y_test, best_rf.predict(X_test_rfe)))
```

Output



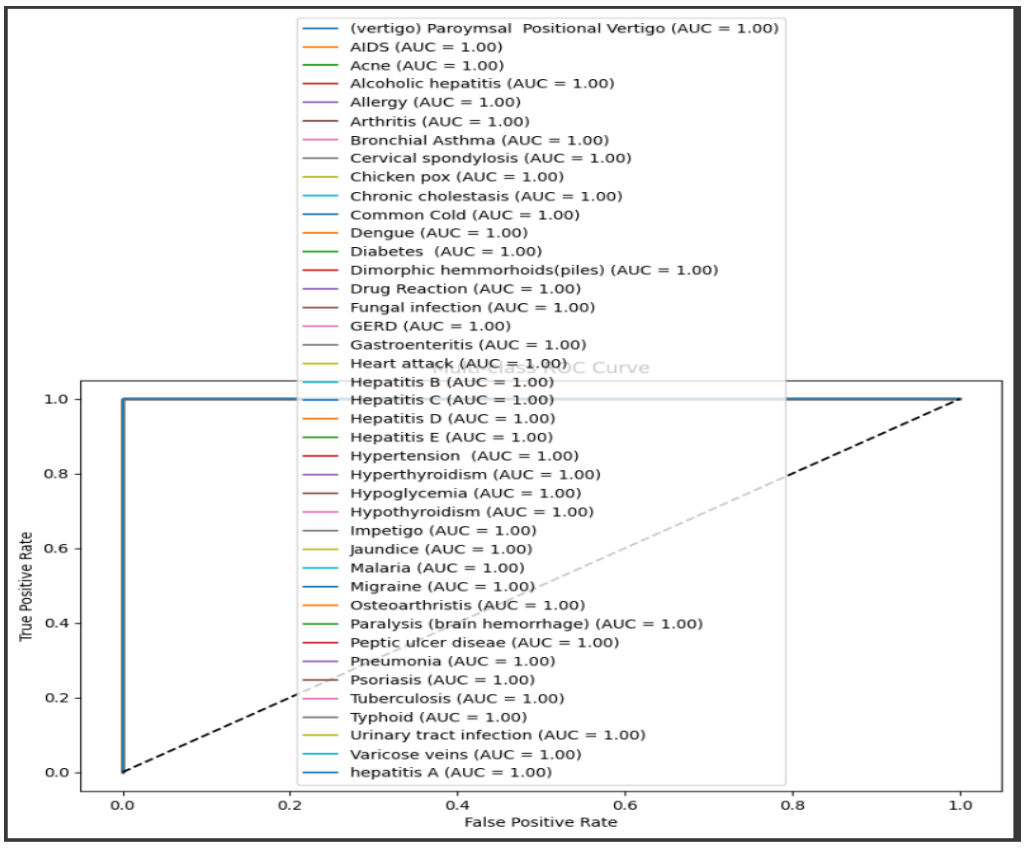
	precision	recall	f1-score	support
0	1.00	1.00	1.00	18
1	1.00	1.00	1.00	30
2	1.00	1.00	1.00	24
3	1.00	1.00	1.00	25
4	1.00	1.00	1.00	24
5	1.00	1.00	1.00	23
6	1.00	1.00	1.00	33
7	1.00	1.00	1.00	23
8	1.00	1.00	1.00	21
9	1.00	1.00	1.00	15
10	1.00	1.00	1.00	23
11	1.00	1.00	1.00	26
12	1.00	1.00	1.00	21
13	1.00	1.00	1.00	29
14	1.00	1.00	1.00	24
15	1.00	1.00	1.00	19
16	1.00	1.00	1.00	28
17	1.00	1.00	1.00	25
18	1.00	1.00	1.00	23
19	1.00	1.00	1.00	27
20	1.00	1.00	1.00	26
21	1.00	1.00	1.00	23
22	1.00	1.00	1.00	29
23	1.00	1.00	1.00	25
24	1.00	1.00	1.00	24
25	1.00	1.00	1.00	26
26	1.00	1.00	1.00	21
27	1.00	1.00	1.00	24
28	1.00	1.00	1.00	19
29	1.00	1.00	1.00	22
30	1.00	1.00	1.00	25
31	1.00	1.00	1.00	22
32	1.00	1.00	1.00	24
33	1.00	1.00	1.00	17
34	1.00	1.00	1.00	28
35	1.00	1.00	1.00	22
36	1.00	1.00	1.00	25
37	1.00	1.00	1.00	19
38	1.00	1.00	1.00	26
39	1.00	1.00	1.00	22
40	1.00	1.00	1.00	34
accuracy			1.00	984
macro avg	1.00	1.00	1.00	984
eighted avg	1.00	1.00	1.00	984

6.3 ROC Curve And AUC (Area Under The Curve)

The True Positive Rate (Recall) is compared to the False Positive Rate on the ROC curve. The model's ability to differentiate between classes is measured by AUC, and the closer it is to 1.0, the better. We employ a One-vs-Rest (OvR) strategy, which generates a distinct ROC curve for each class, since this is a multi-class problem.

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier

# Binarize the output
y_test_bin = label_binarize(y_test, classes=list(range(len(le.classes_))))
y_score = best_rf.predict_proba(X_test_rfe)

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(le.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves
plt.figure(figsize=(10, 6))
for i in range(len(le.classes_)):
    plt.plot(fpr[i], tpr[i], label=f'{le.classes_[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.title('Multi-class ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Output



This visual representation aids in comprehending the behavior of the model across all disease categories and highlights any classes from which the model has difficulty distinguishing

7. Business Insights And Recommendations

7.1 Interpretation Of Model Results

The machine learning models developed in this project, particularly the Random Forest Classifier and Support Vector Machine (SVM), demonstrated strong predictive capabilities in identifying diseases based on patient symptom profiles. Key takeaways include:

High Accuracy and F1-Score: The Random Forest model handled the multi-class classification problem well, with high accuracy across all disease classes.

Recursive Feature Elimination (RFE) and feature importance rankings helped identify the most significant symptoms that contribute to diagnoses. Important features were identified in this.

Effectiveness of Dimensionality Reduction: PCA helped reduce noise and overfitting by reducing feature redundancy while maintaining over 95 percent variance.

7.2 Real World Implications

There are several advantages to using this predictive model in healthcare settings:

Early and Accurate Disease Detection: Using the patient's symptoms and past information, doctors can use this system to get a preliminary diagnosis, allowing for faster treatment.

Decision Support System (DSS): Before prescribing diagnostic tests, the model can serve as a second opinion tool for doctors to narrow down potential illnesses.

Resource Optimization: Hospitals and clinics can better manage resources like lab tests, consultations, and treatments by using the model to prioritize high-risk patients.

Improved Patient Outcomes: Early prediction enables prompt treatment, which can lessen readmission rates, cut down on complications, and speed up recovery.

7.3 Actionable Recommendations

Implement a Web or Mobile Interface: Integrate the trained model into a user-friendly application that patients or healthcare professionals can use in real time. This system can prompt the entry of symptoms and return probable diseases with confidence.

Retrain the Model: Retrain the model on a regular basis with new datasets that reflect new diseases or changes in symptomatology caused by seasonal effects or differences between regions. Stakeholders can be educated by instructing clinical staff and administrators on how to interpret the predictions made by the model and comprehend its limitations. Human judgment should be complemented, not replaced, by the tool.

Privacy and ethics: When collecting and processing patient data, ensure compliance with healthcare regulations like HIPAA and GDPR. Implement robust protocols for anonymization and encryption.

Expand the Dataset's Scope: To increase the model's richness and generalizability, include more diverse data like imaging findings, patient demographics, and lab test results.

8. Limitations

This disease prediction pipeline's generalizability, applicability, and accuracy in real-world clinical settings are hampered by several limitations, despite its promising outcomes:

i. **Binary Symptom Representation:** All symptom features in the dataset are encoded as binary values (0 for absence, 1 for presence). While this makes model training easier and the feature space simpler, it also removes important clinical details:

   There is no indication of severity (such as mild versus severe pain). There is no timing context (such as chronic versus acute symptoms). The model has a harder time distinguishing between diseases with similar symptom sets but varying intensities or durations because of this.

ii. **Synthetic Dataset Constraints:** It appears that the dataset was created synthetically or simulated for academic purposes:

   Here is little clinical variation, which may make patient profiles less diverse and representative. Co-occurring conditions or reporting inconsistencies are examples of real-world anomalies that are not captured. When exposed to clinical records that are noisy and inconsistent, models that were trained on artificial data frequently perform poorly in production.

iii. **Inadequate Temporal Features:** The order of symptom onset, duration, or progression is not considered in the dataset. When diagnosing diseases (such as influenza symptoms versus prolonged COVID), temporal patterns are crucial**.** Models lose important

context, which could otherwise be useful for differential or early diagnosis, without temporal data.

iv. **Unbalanced Classes:** There are far fewer instances of certain diseases than others: Due to a lack of sufficient training data, the model may perform poorly in the treatment of rare diseases. The model is skewed toward predicting conditions that are more prevalent, and recall and precision for rare diseases are reduced as a result.

v. **Insufficient interpretability:** Despite their high accuracy, models like Random Forest and SVM are categorized as "black boxes". To trust and act on model predictions, medical professionals frequently want explanations. It is challenging to justify predictions to healthcare providers in the absence of interpretability tools like SHAP and LIME.

vi. **The inadequacy of Clustering Models:** K-Means and other unsupervised models rely on Euclidean distance-based clustering: This assumption is not well supported by binary symptom features. Low silhouette scores and limited practical segmentation were the results of clustering.

vii. **Overfitting:** There are 134 features in the dataset, but only 4,920 samples. When using complex models, this high feature-to-sample ratio raises the risk of overfitting. While tested well, overfitted models may not be able to apply to new, untested patients.

9.Future Work

To improve performance, realism, and clinical utility of the disease prediction system, several directions can be pursued:

i. **Use of Real-World Patient Data**

Collaborate with hospitals or open-source platforms like MIMIC-III or PhysioNet to obtain de-identified electronic health records (EHRs).

These datasets include richer clinical details like vitals, lab results, medications, and patient history.

Validating the model on EHRs will enhance its reliability for clinical deployment.

ii. **Incorporation of Symptom Severity and Duration**

Modify the dataset or collect data that includes:

Severity scores (1–5 scale)

Symptom duration (in days or weeks)

This would allow the model to differentiate between acute vs. chronic illnesses and improve diagnostic precision.

iii. Exploration of Deep Learning Models

Leverage Recurrent Neural Networks (RNNs) or Transformers to model sequential and contextual symptom data.

These models excel in learning patterns from temporal data and can adapt to variable-length input sequences.

iv. Explainable AI (XAI) Integration

Add explainability modules like:

SHAP (SHapley Additive exPlanations)

LIME (Local Interpretable Model-agnostic Explanations)

These tools help interpret how each symptom contributes to a diagnosis, promoting trust and transparency in healthcare settings.

v. Real-Time Decision Support System

Deploy the model as part of an interactive decision-support system:

Web or mobile app for patients and physicians

Allow real-time symptom entry and receive predictions and recommendations

This can assist in early intervention and triage support in clinics.

vi. Multi-Label Classification

Some patients may present multiple co-occurring conditions.

Extend the classification model to support multi-label classification instead of one-label-per-patient.

This enables more comprehensive diagnostic support in complex cases.

vii. Feature Expansion

Include demographic information (age, gender, ethnicity)

Add lab tests, vitals, medications, and family history

Combining symptoms with other medical data will help the model:

Capture a fuller clinical picture

Improve accuracy and personalization of predictions

viii. Advanced Imputation and Outlier Detection

Implement advanced imputation techniques like:

KNN imputation

Iterative imputation (MICE)

Develop methods to detect and handle suspicious or contradictory symptom combinations that may indicate data entry errors or outliers.

10. Conclusion

A complete end-to-end data science pipeline was created for this project to address the real-world problem of early disease diagnosis with machine learning methods. Beginning with data collection and understanding, the dataset comprising symptoms and corresponding diagnoses was explored through detailed Exploratory Data Analysis (EDA). This phase provided important insights into the frequency and correlation of various clinical features and the distribution of symptoms and target classes. The dataset was cleaned, missing values were handled, the target variable was coded, and features and labels were separated during the subsequent data preprocessing steps. In addition to creating additional informative features like the total symptom count per patient, special care was taken to preserve the integrity of the binary symptom indicators.

The model's capacity to learn relevant patterns was further enhanced by feature engineering. Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are two dimensionality reduction methods that aid in simplifying high-dimensional data into more meaningful subsets, thereby decreasing noise and increasing productivity. Several machine learning models, including the Random Forest Classifier, Support Vector Machine (SVM), and K-Means clustering, were used during the model-building phase. The accuracy, precision, recall, F1-score, confusion matrix, and ROC curves were all used to train and evaluate these models. The Random Forest Classifier emerged as the most effective model with its high predictive accuracy and robustness. Cross-validation and hyperparameter tuning with Grid Search enhanced the models' generalizability, and clustering analysis provided insights into the

grouping behavior of patient profiles, albeit with some drawbacks due to the binary nature of the data. The models developed here could support automated decision systems in clinical settings, particularly primary care or telehealth settings, where early symptom-based predictions can be crucial, which has significant business implications. The identification of key predictive symptoms also provides clinicians with focused diagnostic cues, enhancing efficiency and reducing diagnostic errors.

In general, this project demonstrates how machine learning can be used to improve patient care through intelligent data-driven decision-making and how data science methodologies can be applied to critical healthcare issues. The outcomes underscore the potential for future integration of such systems into real-world healthcare platforms. This pipeline demonstrates the power of data-driven solutions in healthcare.

Finally, through proper EDA, preprocessing, feature engineering, and model tuning, we achieved reliable disease predictions, setting the foundation for real-world applications.