Disease Prediction Using Machine Learning

This article aims to implement a robust machine-learning model that can efficiently predict the disease of a human, based on the symptoms that he/she possesses. Let us look into how we can approach this machine-learning problem

Data Cleaning

In [13]:
```
import pandas as pd
import numpy as np
import pandas as pd
from scipy.stats import mode
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_sco
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

%matplotlib inline
```

In [29]:
```
#disease = pd.read_csv("Training_disease.csv")
DATA_PATH = "Training_disease.csv"
data = pd.read_csv(DATA_PATH).dropna(axis = 1)
```

In [30]:
```
1 data.head(10)
```

Out[30]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint
0	1	1	1	0	0	0	
1	0	1	1	0	0	0	
2	1	0	1	0	0	0	
3	1	1	0	0	0	0	
4	1	1	1	0	0	0	
5	0	1	1	0	0	0	
6	1	0	1	0	0	0	
7	1	1	0	0	0	0	
8	1	1	1	0	0	0	
9	1	1	1	0	0	0	

10 rows × 133 columns

In [31]:
```
1 data.columns
```

Out[31]:
```
Index(['itching', 'skin_rash', 'nodal_skin_eruptions', 'continuous_sne
ezing',
       'shivering', 'chills', 'joint_pain', 'stomach_pain', 'acidity',
       'ulcers_on_tongue',
       ...
       'blackheads', 'scurring', 'skin_peeling', 'silver_like_dustin
g',
       'small_dents_in_nails', 'inflammatory_nails', 'blister',
       'red_sore_around_nose', 'yellow_crust_ooze', 'prognosis'],
      dtype='object', length=133)
```

In [32]:
```
1 data.isnull()
```

Out[32]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	j
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
4915	False	False	False	False	False	False	
4916	False	False	False	False	False	False	
4917	False	False	False	False	False	False	
4918	False	False	False	False	False	False	
4919	False	False	False	False	False	False	

4920 rows × 133 columns

In [33]:
```
1 sum(data["Unnamed: 133"].isnull())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File C:\ProgramData\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3802, in Index.get_loc(self, key, method, tolerance)
   3801 try:
-> 3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:

File C:\ProgramData\anaconda3\lib\site-packages\pandas\_libs\index.pyx:138, in pandas._libs.index.IndexEngine.get_loc()

File C:\ProgramData\anaconda3\lib\site-packages\pandas\_libs\index.pyx:165, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Unnamed: 133'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[33], line 1
----> 1 sum(data["Unnamed: 133"].isnull())

File C:\ProgramData\anaconda3\lib\site-packages\pandas\core\frame.py:3807, in DataFrame.__getitem__(self, key)
   3805 if self.columns.nlevels > 1:
   3806     return self._getitem_multilevel(key)
-> 3807 indexer = self.columns.get_loc(key)
   3808 if is_integer(indexer):
   3809     indexer = [indexer]

File C:\ProgramData\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3804, in Index.get_loc(self, key, method, tolerance)
   3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:
-> 3804     raise KeyError(key) from err
   3805 except TypeError:
   3806     # If we have a listlike key, _check_indexing_error will raise
   3807     #  InvalidIndexError. Otherwise we fall through and re-raise
   3808     #  the TypeError.
   3809     self._check_indexing_error(key)

KeyError: 'Unnamed: 133'
```

In [9]:
```
1 print(disease.isnull().sum())
```

```
itching                   0
skin_rash                 0
nodal_skin_eruptions      0
continuous_sneezing       0
shivering                 0
                        ...
blister                   0
red_sore_around_nose      0
yellow_crust_ooze         0
prognosis                 0
Unnamed: 133           4920
Length: 134, dtype: int64
```

In [34]:
```
1 data = data.dropna()    # Dropping the missing values.
2 data.count()
```

Out[34]:
```
itching                4920
skin_rash              4920
nodal_skin_eruptions   4920
continuous_sneezing    4920
shivering              4920
                       ...
inflammatory_nails     4920
blister                4920
red_sore_around_nose   4920
yellow_crust_ooze      4920
prognosis              4920
Length: 133, dtype: int64
```

In [35]:
```
1 print(data.isnull().sum())
```

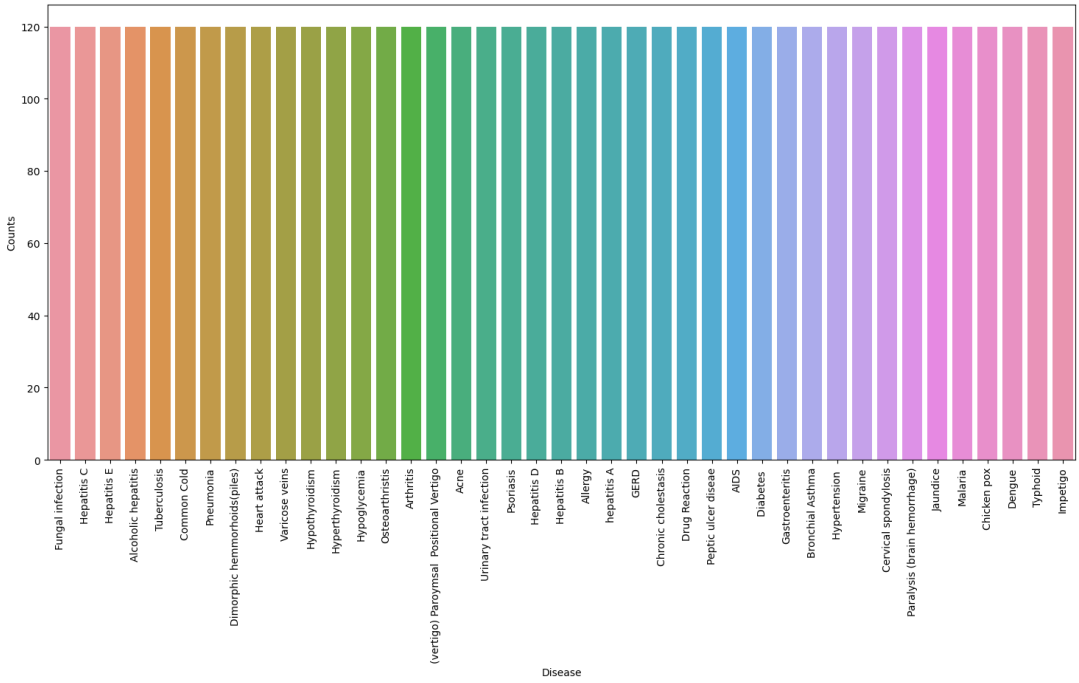
```
itching                0
skin_rash              0
nodal_skin_eruptions   0
continuous_sneezing    0
shivering              0
                      ..
inflammatory_nails     0
blister                0
red_sore_around_nose   0
yellow_crust_ooze      0
prognosis              0
Length: 133, dtype: int64
```

In [36]:
```
1 duplicate_rows_data = data[data.duplicated()]
2 print("number of duplicate rows: ", duplicate_rows_data.shape)
```

```
number of duplicate rows:  (4616, 133)
```

In [37]:
```
# 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
})

```

In [38]:
```
plt.figure(figsize = (18,8))
sns.barplot(x = "Disease", y = "Counts", data = temp_df)
plt.xticks(rotation=90)
plt.show()
```



Encoding the target value into numerical

value using LabelEncoder

In [39]:
```
encoder = LabelEncoder()
data["prognosis"] = encoder.fit_transform(data["prognosis"])
```

Splitting the data for training and testing the model

In [40]:
```
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}")

```

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

Using K-Fold Cross-Validation for model selection

In [41]:
```
# 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)}")
```

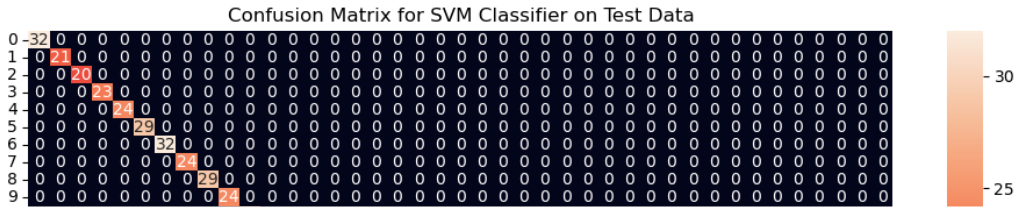
```
============================================================
SVC
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================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

Building robust classifier by combining all models:

In [42]:

```
# 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 Da
plt.show()

```

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

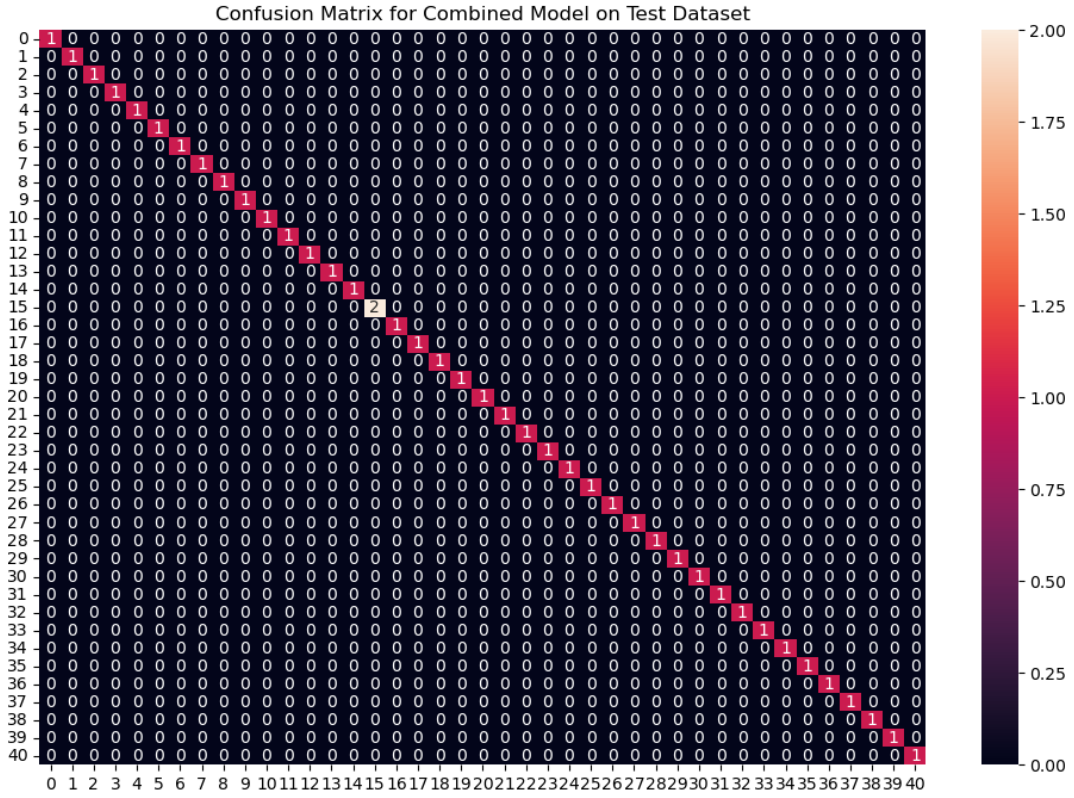


Fitting the model on whole data and validating on the Test dataset:

In [44]:

```
# Training the models on whole data
final_svm_model = SVC()
final_nb_model = GaussianNB()
final_rf_model = RandomForestClassifier(random_state=18)
final_svm_model.fit(X, y)
final_nb_model.fit(X, y)
final_rf_model.fit(X, y)

# Reading the test data
test_data = pd.read_csv("Testing_disease.csv").dropna(axis=1)

test_X = test_data.iloc[:, :-1]
test_Y = encoder.transform(test_data.iloc[:, -1])

# Making prediction by take mode of predictions
# made by all the classifiers
svm_preds = final_svm_model.predict(test_X)
nb_preds = final_nb_model.predict(test_X)
rf_preds = final_rf_model.predict(test_X)

final_preds = [mode([i,j,k])[0][0] for i,j,
            k in zip(svm_preds, nb_preds, rf_preds)]

print(f"Accuracy on Test dataset by the combined model\
: {accuracy_score(test_Y, final_preds)*100}")

cf_matrix = confusion_matrix(test_Y, final_preds)
plt.figure(figsize=(12,8))

sns.heatmap(cf_matrix, annot = True)
plt.title("Confusion Matrix for Combined Model on Test Dataset")
plt.show()

```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_19396\408761218.py:21: Future
Warning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), t
he default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `
keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted.
Set `keepdims` to True or False to avoid this warning.
  final_preds = [mode([i,j,k])[0][0] for i,j,

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


Confusion Matrix for Combined Model on Test Dataset
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

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

In [45]:

```
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
    nb_prediction = data_dict["predictions_classes"][final_nb_model
    svm_prediction = data_dict["predictions_classes"][final_svm_mod

    # making final prediction by taking mode of all predictions
    final_prediction = mode([rf_prediction, nb_prediction, svm_pred
    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"))
```

```
{'rf_model_prediction': 'Fungal infection', 'naive_bayes_prediction':
'Fungal infection', 'svm_model_prediction': 'Fungal infection', 'final
_prediction': 'Fungal infection'}
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWa
rning: X does not have valid feature names, but RandomForestClassifier
was fitted with feature names
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWa
rning: X does not have valid feature names, but GaussianNB was fitted
with feature names
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWa
rning: X does not have valid feature names, but SVC was fitted with fe
ature names
  warnings.warn(
C:\Users\hp\AppData\Local\Temp\ipykernel_19396\2615708464.py:37: Futur
eWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`),
the default behavior of `mode` typically preserves the axis it acts al
ong. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepte
d. Set `keepdims` to True or False to avoid this warning.
  final_prediction = mode([rf_prediction, nb_prediction, svm_predictio
n])[0][0]
C:\Users\hp\AppData\Local\Temp\ipykernel_19396\2615708464.py:37: Depre
cationWarning: Support for non-numeric arrays has been deprecated as o
f SciPy 1.9.0 and will be removed in 1.11.0. `pandas.DataFrame.mode` c
an be used instead, see https://pandas.pydata.org/docs/reference/api/p
andas.DataFrame.mode.html. (https://pandas.pydata.org/docs/reference/a
pi/pandas.DataFrame.mode.html.)
  final_prediction = mode([rf_prediction, nb_prediction, svm_predictio
n])[0][0]
```

Observation :- The symptoms that are given as input to the function should be exactly the same among the 132 symptoms in the dataset.

In [ ]:

```
1
```