Coding:

Accuracy for modelling Techniques for SVM algorithm:

```
import NumPy as np
import pandas as pd
import matplotlib. pyplot as plt
import seaborn as sns
##
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_scor
from sklearn.model_selection import GridSearchCV
from sklearn.utils import shuffle
from sklearn import metrics
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score
## import data
df = pd.read_csv('/content/trained.csv')
df = shuffle (df, random_state = 42)
df.head()
## remove ('_') underscore in the text
for col in df.columns:
    df[col] = df[col].str.replace('_', ' ')
df.head()
## charactieristics of data
df.describe()
## check null values
null_checker=df. apply(lambdax:sum
(x.isnull())).to_frame(name='count')
print(null_checker)
## plot of null value
plt.figure(figsize=(10, 5), dpi=140)
plt.plot(null_checker.index, null_checker['count'])
plt.xticks(null_checker.index,null_checker.index,rotation=45,horizo
ntalalignment = 'right')
plt.title('Ratio of Null values')
plt.xlabel('column names')
plt.margins(0.1)
plt.show()
cols=df.columns
data= df[cols].values.flatten()
reshaped = pd.Series(data)
reshaped = reshaped.str.strip()
reshaped = reshaped.values.reshape(df.shape)
```

```
df = pd.DataFrame(reshaped, columns = df.columns)
df.head()
## split data
data = df_processed.iloc[:,1:].values
labels = df['Disease']. values
X_train, X_test, y_train, y_test = train_test_split(data,labels,
 test_size= 0.2, random_state=42)
 def performance_evaluator(model, X_test, y_test):
    """
    model: Load the trained model
    X_test: test data
    y_test: Actual value

    """
    y_predicted = model.predict(X_test)
    precision = precision_score(y_test,
y_predicted,average='micro')*100
    accuracy = accuracy_score(y_test, y_predicted)*100
    f1 = f1_score (y_test, y_predicted, average='macro')*100
    recall = recall_score(y_test, y_predicted, average='macro')*100
    print('precision----->', precision)
    print('\n************************')
    print('Accuracy----->', accuracy)
    print('\n************************')
    print ('F1 Score----->', f1)
    print('\n************************')
    print('Recall----->', recall)
    print('\n************************')
    return accuracy, precision, f1, recall
## plot classification metrix
   def confusion_plot(model, X_test, y_test):
    """
    to plot confusion metrix
    """
    plt.figure(figsize=(10, 10), dpi=150)
    y_pred = model.predict(X_test)
    con_me = confusion_matrix(y_test, y_pred)
    sns.heatmap(con_me, annot=True)
    SVM_init = SVC ()
  model_SVM_init = SVM_init.fit(X_train, y_train)
_1,    _2,_3,_4=performance_evaluator(model_SVM_init,    X_test,
y_test)
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01,
0.001, 0.0001], 'kernel': ['rbf']}
 grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
grid.fit(X_train, y_train)
 print(grid.best_estimator_)
print(grid.best_params_)
## lets built based SVC model.
hyper_tuned_svc = SVC (C= 10, gamma= 0.1, kernel= 'rbf')
```

```
hyper_tuned_svc.fit(X_train, y_train)
_1, _2, _3, _4 = performance_evaluator(hyper_tuned_svc,
  X_test,y_test)
confusion_plot(hyper_tuned_svc, X_test, y_test)
```

Model Building

```
from tkinter import *
import numpy as np
import pandas as pd
# from gui_stuff import *
l1=['back_pain','constipation','abdominal_pain','diarrhoea','mild_fever',
'yellow_urine','yellowing_of_eyes','acute_liver_failure','fluid_overload',
'swelling_of_stomach','swelled_lymph_nodes','malaise','blurred_and_distorted_vision','phlegm','redness_of_eyes','sinus_pressure','runny_nose','congestion','chest_pain','throat_irritation','weakness_in_limbs','fast_heart_rate','pain_during_bowel_movements','pain_in_anal_region','bloody_stool', 'irritation_in_anus', 'neck_pain', 'dizziness','cramps', 'bruising', 'obesity', 'swollen_legs', 'swollen_blood_vessels','puffy_face_and_eyes',
'enlarged_thyroid','brittle_nails','swollen_extremeties',
'excessive_hunger','extra_marital_contacts','drying_and_tingling_lips',
'slurred_speech','knee_pain','hip_joint_pain','muscle_weakness','stiff_neck','swelling_joints','movement_stiffness','spinning_movements', 'loss_of_balance',
'unsteadiness','weakness_of_one_body_side','loss_of_smell',
'bladder_discomfort','foul_smell_of_urine','continuous_feel_of_urine','passage_of_gases','internal_itching','toxic_look_(typhos)','depression','irritability','muscle_pain','altered_sensorium','red_spots_over_body','belly_pain','abnormal_menstruation','dischromic_patches','watering_from_eyes','increased_appetite','polyuria','family_history','mucoid_sputum','rusty_sputum','lack_of_concentration','visual_disturbances',
'receiving_blood_transfusion','receiving_unsterile_injections','coma','stomach_bleeding','distention_of_abdomen','history_of_alcohol_consumption','fluid_overload','blood_in_sputum','prominent_veins_on_calf',
'palpitations','painful_walking','pus_filled_pimples','blackheads',
'scurring','skin_peeling','silver_like_dusting','small_dents_in_nails','inflammatory_nails','blister','red_sore_around_nose','yellow_crust_ooze'])
 disease = ['Fungal infection', 'Allergy', 'GERD', 'Chronic cholestasis',
'DrugReaction','Pepticulcerdiseases','AIDS','Diabetes','Gastroenteritis','BronchialAsthma','Hypertension','Migraine','Cervicalspondylosis','Paralysis (brain hemorrhage)', 'Hepatitis B', 'Hepatitis C', 'Hepatitis D', 'Hepatitis E', 'Alcoholic hepatitis','Chicken pox', 'Dengue', 'Typhoid', 'hepatitis A',
```

```
'Jaundice’,‘Malaria’,'Tuberculosis','CommonCold','Pneumonia','Dim
orphichemmorhoids(piles)','Heartattack','Varicoseveins','Hypothyroi
dism','Hyperthyroidism','Hypoglycemia', 'Osteoarthristis','Arthritis',
'(vertigo) ParoxysmalPositional’])
l2 = []
for x in range (0, len(l1)):
   l2. append (0)
dataset=pd.read_csv (r"D:\loki\backup project\New_Projects
Backup\30. Symptom based Disease prediction using Machine
Learning\SOURCE CODE/Training.csv")
dataset.replace({'prognosis': {'Fungal infection': 0, 'Allergy': 1,
'GERD': 2, 'Chronic cholestasis': 3, 'Drug Reaction': 4,'Peptic ulcer
diseases': 5, 'AIDS': 6, 'Diabetes ': 7, 'Gastroenteritis': 8, 'Bronchial
Asthma': 9, 'Hypertension ': 10,'Migraine': 11, 'Cervical spondylosis':
12,'Paralysis (brain hemorrhage)': 13, 'Jaundice': 14, 'Malaria': 15,
'Chicken pox': 16, 'Dengue': 17, 'Typhoid': 18, 'hepatitis A':
19,'Hepatitis B': 20, 'Hepatitis C': 21, 'Hepatitis D': 22, 'Hepatitis E':
23, 'Alcoholic hepatitis': 24, 'Tuberculosis': 25,'Common Cold': 26,
'Pneumonia': 27, 'Dimorphic hemmorhoids(piles)': 28, 'Heart attack':
29, 'Varicose veins': 30, 'Hypothyroidism': 31,'Hyperthyroidism': 32,
'Hypoglycemia': 33, 'Osteoarthristis': 34, 'Arthritis': 35,'(vertigo)
Paroxysmal Positional Vertigo': 36, 'Acne': 37, 'Urinary tract
infection': 38, 'Psoriasis': 39,'Impetigo': 40}}, inplace=True)
print(dataset.head())
X = dataset[l1]
y = dataset[["prognosis"]]
np.ravel(y)
print(y)
df_test= pd.read_csv (r"D:\loki\backup project\New_Projects
Backup\30. Symptom based Disease prediction using Machine
Learning\SOURCE CODE/Testing.csv")
df_test.replace({'prognosis': {'Fungal infection': 0, 'Allergy': 1,
'GERD': 2, 'Chronic cholestasis': 3, 'Drug Reaction': 4, 'Peptic ulcer
diseae': 5, 'AIDS': 6, 'Diabetes ': 7, 'Gastroenteritis': 8, 'Bronchial
Asthma': 9, 'Hypertension ': 10, 'Migraine': 11, 'Cervical spondylosis':
12, 'Paralysis
(brain hemorrhage)': 13, 'Jaundice': 14, 'Malaria': 15, 'Chicken pox':
16, 'Dengue': 17, 'Typhoid': 18, 'hepatitis A': 19, 'Hepatitis B': 20,
'Hepatitis C': 21, 'Hepatitis D': 22, 'Hepatitis E': 23, 'Alcoholic
hepatitis': 24, 'Tuberculosis': 25, 'Common Cold': 26, 'Pneumonia':
27, 'Dimorphic hemmorhoids(piles)': 28, 'Heart attack': 29, 'Varicose
veins': 30, 'Hypothyroidism': 31, 'Hyperthyroidism': 32,
'Hypoglycemia': 33, 'Osteoarthristis': 34, 'Arthritis': 35, '(vertigo)
Paroymsal  Positional Vertigo': 36, 'Acne': 37, 'Urinary tract
infection': 38, 'Psoriasis': 39,'Impetigo': 40}}, inplace=True)
X_test = df_test[l1]
y_test = df_test[["prognosis"]]
np.ravel(y_test)
def DecisionTree():
 from sklearn import tree
```

```
    DT = tree.DecisionTreeClassifier()  # empty model of the decision
tree
    DT = DT.fit(X, y)
from sklearn.metrics import accuracy_score
 y_pred = DT.predict(X_test)
 print(accuracy_score(y_test, y_pred))
 print(accuracy_score(y_test, y_pred, normalize=False))
psymptom=[Symptom1.get(),Symptom2.get(),Symptom3.get(),Sym
ptom4.
get(), Symptom5.get()]
   for k in range(0, len(l1)):
        for z in psymptoms:
            if(z == l1[k]):
                l2[k] = 1
    inputtest = [l2]
    predict = DT.predict(inputtest)
    predicted = predict[0]
    h = 'no'
    for a in range(0, len(disease)):
    if(predicted == a):
            h = 'yes'
            break
    if (h == 'yes'):
        t1.delete("1.0", END)
        t1.insert(END, disease[a])
    else:
        t1.delete("1.0", END)
        t1.insert(END, "Not Found")
def randomforest():
    from sklearn.ensemble import RandomForestClassifier
    Rf = RandomForestClassifier()
    Rf = Rf.fit(X, np.ravel(y))
from sklearn.metrics import accuracy_score
    y_pred = Rf.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred, normalize=False))
     psymptoms = [Symptom1.get (), Symptom2.get (), Symptom3.get
(),   Symptom4.get (), Symptom5.get ()]
for k in range(0, len(l1)):
        for z in psymptoms:
            if(z == l1[k]):
                l2[k] = 1
    inputtest = [l2]
    predict = Rf.predict(inputtest)
    predicted = predict[0]
    h = 'no'
    for a in range(0, len(disease)):
        if(predicted == a):
            h = 'yes'
            break
```

```
    if (h == 'yes'):
        t2.delete("1.0", END)
        t2.insert(END, disease[a])
    else:
        t2.delete("1.0", END)
        t2.insert(END, "Not Found")
def NaiveBayes():
    from sklearn.naive_bayes import GaussianNB
    gnb = GaussianNB()
    gnb = gnb.fit(X, np.ravel(y))
    from sklearn.metrics import accuracy_score
    y_pred = gnb.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred, normalize=False))
    psymptoms = [Symptom1.get(), Symptom2.get(),
Symptom3.get(),
     Symptom4.get(), Symptom5.get()]
    for k in range(0, len(l1)):
        for z in psymptoms:
            if(z == l1[k]):
                l2[k] = 1
    inputtest = [l2]
    predict = gnb.predict(inputtest)
    predicted = predict[0]
    h = 'no'
    for a in range(0, len(disease)):
        if(predicted == a):
            h = 'yes'
            break
    if (h == 'yes'):
        t3.delete("1.0", END)
        t3.insert(END, disease[a])
    else:
        t3.delete("1.0", END)
        t3.insert(END, "Not Found")
root = Tk()
root.configure(background='black')
Symptom1 = StringVar()
Symptom1.set(None)
Symptom2 = StringVar()
Symptom2.set(None)
Symptom3 = StringVar()
Symptom3.set(None)
Symptom4 = StringVar()
Symptom4.set(None)
Symptom5 = StringVar()
Symptom5.set(None)
Name = StringVar()
w2 = Label(root, justify=LEFT, text="Disease Predictor”,
fg="white", bg="black")
```

```
w2.config(font=("Elephant", 30))
w2.grid(row=1, column=0, columnspan=2, padx=100)
w2.config(font=("Aharoni", 30))
w2.grid(row=2, column=0, columnspan=2, padx=100)
NameLb = Label(root, text="Name of the Patient", fg="yellow", bg="black")
NameLb.grid(row=6, column=0, pady=10, sticky=W)
S1Lb = Label(root, text="Symptom 1", fg="yellow", bg="black",)
S1Lb.grid(row=7, column=0, pady=10, sticky=W)
S2Lb = Label(root, text="Symptom 2", fg="yellow", bg="black")
S2Lb.grid(row=8, column=0, pady=10, sticky=W)
S3Lb = Label(root, text="Symptom 3", fg="yellow", bg="black")
S3Lb.grid(row=9, column=0, pady=10, sticky=W)
S4Lb = Label(root, text="Symptom 4", fg="yellow", bg="black")
S4Lb.grid(row=10, column=0, pady=10, sticky=W)
S5Lb = Label(root, text="Symptom 5", fg="yellow", bg="black")
S5Lb.grid(row=11, column=0, pady=10, sticky=W)
lrLb = Label(root, text="Decision Tree", fg="white", bg="red")
lrLb.grid(row=15, column=0, pady=10, sticky=W)
destreeLb=Label(root,text="RandomForest Tree", fg="white", bg="red")
destreeLb.grid(row=17, column=0, pady=10, sticky=W)
ranfLb = Label(root, text="Naive Bayes", fg="white", bg="red")
ranfLb.grid(row=19, column=0, pady=10, sticky=W)
OPTIONS = sorted(l1)
NameEn = Entry(root, textvariable=Name)
NameEn.grid(row=6, column=1)
S1En = OptionMenu(root, Symptom1, *OPTIONS)
S1En.grid(row=7, column=1)
S2En = OptionMenu(root, Symptom2, *OPTIONS)
S2En.grid(row=8, column=1)
S3En = OptionMenu(root, Symptom3, *OPTIONS)
S3En.grid(row=9, column=1)
S4En = OptionMenu(root, Symptom4, *OPTIONS)
S4En.grid(row=10, column=1)
S5En = OptionMenu(root, Symptom5, *OPTIONS)
S5En.grid(row=11, column=1)
dst=Button(root,text="DecisionTree",command=DecisionTree,
bg="green", fg="yellow")
dst.grid(row=8, column=3, padx=10)
rnf=Button(root,text="RandomForestTree",
command=randomforest, bg="green", fg="yellow")
rnf.grid(row=9, column=3, padx=10)
lr = Button (root, text="Naive Bayes", command=NaiveBayes, bg="green", fg="yellow")
lr.grid(row=10, column=3, padx=10)
t1 = Text(root, height=1, width=40, bg="orange", fg="black")
t1.grid(row=15, column=1, padx=10)
t2 = Text(root, height=1, width=40, bg="orange", fg="black")
t2.grid(row=17, column=1, padx=10)
```

```
t3 = Text(root, height=1, width=40, bg="orange", fg="black")
t3.grid(row=19, column=1, padx=10)
root.mainloop()
```