Source Code:

```
# -*- coding: utf-8 -*-
"""PulmonaryDisease_Classification.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Xfo0aHrgs49rpUvTICGg584zXDcWulIi
"""

import pandas as pd
import numpy as nm
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import KFold
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.impute import KNNImputer
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets, linear_model, metrics
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

from  google.colab import files
upload= files.upload()

dataset=pd.read_csv('PulmonaryDisease5.csv')
d2=pd.read_csv('123.csv')

m=d2.corr()
plt.figure(figsize=(35,25))
sn.set(font_scale=1.8)
sn.heatmap(m, annot=True,cmap='rocket',annot_kws={"size":18})
```

```
plt.show()

from sklearn.impute import KNNImputer
imputer=KNNImputer(n_neighbors=10)
dataset= pd.DataFrame(imputer.fit_transform(dataset),columns=dataset.columns)

dataset.columns

y=dataset[['cough', 'breathlessness', 'headache', 'mild_fever',
       'throat_irritation', 'runny_nose', 'sinus_pressure',
       'chest_pain', 'blood_in_sputum']]

y.shape

x=dataset['prognosis']

x.shape

y.shape

X_train, X_test, y_train, y_test = train_test_split(
    y, x, test_size=0.2, random_state=42
)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

"""User Defined Functions"""

def ResultPrint(acc, prec, recall, f1, model_name):
  print('Pulmonary Disease Categorization:\n')
  print('Model:',model_name,'\nAccuracy =', format(ac,".4f"),
      '\nPrecision=',format(prec, ".4f"),'\nRecall=', format(rec, ".4f"),'\nF1
Score=',format(f1score,".4f"))

"""Function for Cross Value Score"""

import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn import linear_model, tree, ensemble

kf =KFold(n_splits=10, shuffle=True, random_state=42)

cnt=1
```

```
for train_index, test_index in kf.split(y, x):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:{len(test_index)}')
    cnt += 1

import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.model_selection import validation_curve

"""Classification Models"""

model = RandomForestClassifier(random_state=0)
model.fit(X_train, y_train)
preds = model.predict(X_test)
prec = precision_score(y_test, preds, average='macro')
f1score = f1_score(y_test, preds, average='macro')
rec=recall_score(y_test, preds, average='macro')
ac=accuracy_score(y_test, preds)
ResultPrint(ac,prec,rec,f1score,"Random Forest Classifier")

acc= cross_val_score(RandomForestClassifier(random_state= 42, criterion='entropy'),
y, x, cv=kf, scoring="accuracy")
prec= cross_val_score(RandomForestClassifier(random_state= 42), y, x, cv=kf,
scoring="precision_macro")
recall= cross_val_score(RandomForestClassifier(random_state= 42), y, x, cv=kf,
scoring="recall_macro")
f1= cross_val_score(RandomForestClassifier(random_state= 42), y, x, cv=kf,
scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Random Forest Classifier")

"""Parameter Optimization"""

n_estimators = [5,20,50,100] # number of trees in the random forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)] # maximum number of
levels allowed in each decision tree
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf
node
```

```
bootstrap = [True, False] # method used to sample data points

random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}

## Importing Random Forest Classifier from the sklearn.ensemble
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestClassifier()

from sklearn.model_selection import RandomizedSearchCV
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 100, cv = 5, verbose=2, random_state=35, n_jobs = -1)

rf_random.fit(X_train, y_train)

rf_random.best_params_

acc= cross_val_score(RandomForestClassifier(bootstrap=False,max_depth=110,
max_features='auto', min_samples_leaf=1, min_samples_split=6, n_estimators=50), y,
x, cv=kf, scoring="accuracy")
prec= cross_val_score(RandomForestClassifier(bootstrap=False,max_depth=110,
max_features='auto', min_samples_leaf=1, min_samples_split=6, n_estimators=50), y,
x, cv=kf, scoring="precision_macro")
recall= cross_val_score(RandomForestClassifier(bootstrap=False,max_depth=110,
max_features='auto', min_samples_leaf=1, min_samples_split=6, n_estimators=50), y,
x, cv=kf, scoring="recall_macro")
f1= cross_val_score(RandomForestClassifier(bootstrap=False,max_depth=110,
max_features='auto', min_samples_leaf=1, min_samples_split=6, n_estimators=50), y,
x, cv=kf, scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Random Forest Classifier")

"""Decision Tree Classifier"""
```

```
model = DecisionTreeClassifier(criterion='gini',random_state=0)
model.fit(X_train, y_train)
preds = model.predict(X_test)
prec = precision_score(y_test, preds, average='macro')
f1score = f1_score(y_test, preds, average='macro')
rec=recall_score(y_test, preds, average='macro')
ac=accuracy_score(y_test, preds)
ResultPrint(ac,prec,rec,f1score,"Decision Tree Classifier")

acc= cross_val_score(DecisionTreeClassifier(random_state= 42, criterion='entropy'),
y, x, cv=kf, scoring="accuracy")
prec= cross_val_score(DecisionTreeClassifier(random_state= 42), y, x, cv=kf,
scoring="precision_macro")
recall= cross_val_score(DecisionTreeClassifier(random_state= 42), y, x, cv=kf,
scoring="recall_macro")
f1= cross_val_score(DecisionTreeClassifier(random_state= 42), y, x, cv=kf,
scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Decision Tree Classifier")

"""Decision Tree with parameter Optimization

"""

from sklearn.model_selection import GridSearchCV
params = {
    'max_depth': [3,5,8,12,15],
    'min_samples_leaf': [5,8,12,15,20],
    'criterion': ["gini", "entropy"]
}
dt=DecisionTreeClassifier(random_state=42)

grid_search = RandomizedSearchCV(estimator=dt, param_distributions = params, n_iter
= 100, cv = 5, verbose=2, random_state=35, n_jobs = -1)

grid_search.fit(X_train, y_train)

grid_search.best_estimator_

acc= cross_val_score(DecisionTreeClassifier(max_depth=5, min_samples_leaf=5,
random_state=42), y, x, cv=kf, scoring="accuracy")
```

```
prec= cross_val_score(DecisionTreeClassifier(max_depth=5, min_samples_leaf=5,
random_state=42), y, x, cv=kf, scoring="precision_macro")
recall= cross_val_score(DecisionTreeClassifier(max_depth=5, min_samples_leaf=5,
random_state=42), y, x, cv=kf, scoring="recall_macro")
f1= cross_val_score(DecisionTreeClassifier(max_depth=5, min_samples_leaf=5,
random_state=42), y, x, cv=kf, scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Decision Tree Classifier")

"""Gradient Boost"""

model=GradientBoostingClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
prec = precision_score(y_test, preds, average='macro')
f1score = f1_score(y_test, preds, average='macro')
rec=recall_score(y_test, preds, average='macro')
ac=accuracy_score(y_test, preds)

print('Pulmonary Disease Categorization:\n')
print('Model: Gradient Boosting Classifier','\nAccuracy =', format(ac,".4f"),
      '\nPrecision=',format(prec, ".4f"),'\nRecall=', format(rec, ".4f"),'\nF1
Score=',format(f1score,".4f"))



acc= cross_val_score(GradientBoostingClassifier(), y, x, cv=kf, scoring="accuracy")
prec= cross_val_score(GradientBoostingClassifier(), y, x, cv=kf,
scoring="precision_macro")
recall= cross_val_score(GradientBoostingClassifier(), y, x, cv=kf,
scoring="recall_macro")
f1= cross_val_score(GradientBoostingClassifier(), y, x, cv=kf, scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Gradient Boosting Classifier")

"""Parameter Optimization"""

parameters = {
    "loss":["deviance"],
```

```
    "learning_rate": [0.01, 0.025, 0.075, 0.15],
    "min_samples_split": np.linspace(0.1, 0.5),
    "min_samples_leaf": np.linspace(0.1, 0.5),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],

    "subsample":[0.5,   1.0],
    "n_estimators":[10]
    }

from sklearn.model_selection import GridSearchCV
clf = RandomizedSearchCV(estimator=GradientBoostingClassifier(), param_distributions
= parameters, n_iter = 100, cv = 5, verbose=2, random_state=35, n_jobs = -1)

clf.fit(X_train, y_train)
#converting the clf.cv_results to dataframe

clf.best_params_

model=SVC(kernel='linear')
model.fit(X_train, y_train)
preds = model.predict(X_test)
prec = precision_score(y_test, preds, average='macro')
f1score = f1_score(y_test, preds, average='macro')
rec=recall_score(y_test, preds, average='macro')
ac=accuracy_score(y_test, preds)

print('Pulmonary Disease Categorization:\n')
print('Model: Support Vector Machine Classifier','\nAccuracy =', format(ac,".4f"),
      '\nPrecision=',format(prec, ".4f"),'\nRecall=', format(rec, ".4f"),'\nF1
Score=',format(f1score,".4f"))

acc= cross_val_score(GradientBoostingClassifier(subsample=1.0,
 n_estimators=10,
 min_samples_split=0.3857142857142858,
 min_samples_leaf=0.1163265306122449,
 max_features='log2',
 max_depth=3,
 loss='deviance',
 learning_rate= 0.075), y, x, cv=kf, scoring="accuracy")
prec= cross_val_score(SVC(random_state= 42), y, x, cv=kf, scoring="precision_macro")
recall= cross_val_score(SVC(random_state= 42), y, x, cv=kf, scoring="recall_macro")
f1= cross_val_score(SVC(random_state= 42), y, x, cv=kf, scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
```

```
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Support Vector Machine Classifier")

start = timer()
end = timer()
print("\n")
print(end - start)

"""Parameter Optimization"""

param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'linear']}

grid = RandomizedSearchCV(SVC(), param_grid, refit = True, verbose = 3)
grid.fit(X_train, y_train)

grid.best_params_

acc= cross_val_score(SVC(kernel='rbf', gamma=0.1, C=1), y, x, cv=kf,
scoring="accuracy")
prec= cross_val_score(SVC(random_state= 42, kernel='rbf', gamma=0.1, C=1), y, x,
cv=kf, scoring="precision_macro")
recall= cross_val_score(SVC(random_state= 42, kernel='rbf', gamma=0.1, C=1), y, x,
cv=kf, scoring="recall_macro")
f1= cross_val_score(SVC(random_state= 42, kernel='rbf', gamma=0.1, C=1), y, x,
cv=kf, scoring="f1_macro")
a=np.mean(acc)
p=np.mean(prec)
r=np.mean(recall)
f=np.mean(f1)
ResultPrint(a,p,r,f,"Support Vector Machine Classifier")
from timeit import default_timer as timer

start = timer()
end = timer()
print("\n")
print(end - start)

"""Artificial Neural Network"""
import random
from sklearn.utils import shuffle
from sklearn import metrics
from sklearn.metrics import cohen_kappa_score
```

```
#%%                    Import Dataset
dataset = pd.read_csv('data.csv', encoding= 'unicode_escape')
#print(dataset.head(5))
#dataset = shuffle(dataset)
print("Data shape:",dataset.shape)

# iterating the columns
for col in dataset.columns:
    print(col)

#%% Missing value

from sklearn.impute import KNNImputer
imputer=KNNImputer(n_neighbors=5)
dataset= pd.DataFrame(imputer.fit_transform(dataset),columns=dataset.columns)

#%%                Create X and Y variables

X=dataset[['cough', 'breathlessness', 'headache', 'mild_fever',
       'throat_irritation', 'runny_nose', 'sinus_pressure',
       'chest_pain', 'blood_in_sputum']]



y=dataset['prognosis']


#%%                    Feature Correlation

import seaborn as sn
import matplotlib.pyplot as plt
sn.set(font_scale=1.8)
sn.set_style("darkgrid")
fig_dims = (20, 12)
fig, ax = plt.subplots(figsize=fig_dims)
sn.heatmap(dataset.corr(),annot=True, ax=ax)
plt.show()



#%%            Spliting the dataset into Training set and Test set

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

#%%                         Feature Scaling
'''
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
'''

#%%------------------------- Train The Model-----------------------


#%%                  Initialize the model

# ANN
#%% Functions

# Plot
def plott_ann(train,val,x_test,x_val,x):
    import seaborn as sn
    import matplotlib.pyplot as plt

    sn.set(font_scale=4)
    plt.rcParams['figure.figsize']=20,10
    sn.set_style("whitegrid")
    plt.plot(train,linewidth=5, label  = x_test)
    plt.plot(val, linewidth=5,  label  = x_val)
    plt.xlabel('epoch',fontsize = 50)
    plt.ylabel(x,fontsize = 50 )
    plt.grid(True)
    plt.legend()
    #plt.title("Model Performance", fontsize = 20)
    plt.rc('xtick', labelsize=30)
    plt.rc('ytick', labelsize=30)
    plt.legend(loc=5, prop={'size': 30})
    plt.show()

# model score function
def model_score (y_test,y_pred):
    # Statistical Score
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
    st = 'macro'  # 'micro',  'macro' , 'weighted' None
    ac  = accuracy_score(y_test, y_pred)  # for multy class
    pre = precision_score(y_test, y_pred, average = st) # use [accuracy_score(y_test, y_pred, average = st)]
```

```
    re  = recall_score(y_test, y_pred, average = st)
    f1  = f1_score(y_test, y_pred, average = st)
    results = [ac,pre,re, f1]
    return results

# print function
def print_summary(ac,pr,re,f):
    print("Accuracy    =", ac)
    print("Precision   =", pr)
    print("Recall      =", re)
    print("F1 Score    =", f)


#%%  Importing the Keras libraries and packages

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

#%%                    Create the Model

# Adding the input layer and the first hidden layer
classifier.add(Dense( 30, input_dim = 9,  activation = 'relu' ))

# # Adding the input layer and the first hidden layer
classifier.add(Dense( 12, input_dim = 30,  activation = 'relu' ))

# # Adding the second hidden layer
# classifier.add(Dense( 16, input_dim = 32, activation = 'relu'))

# # Adding the input layer and the first hidden layer
# classifier.add(Dense( 8, input_dim = 16,  activation = 'relu' ))

# # Adding the input layer and the first hidden layer
# classifier.add(Dense( 6, input_dim = 3,  activation = 'relu' ))

# # Adding the input layer and the first hidden layer
# classifier.add(Dense( 12, input_dim = 6,  activation = 'relu' ))

# # Adding the input layer and the first hidden layer
# classifier.add(Dense( 18, input_dim = 12,  activation = 'relu' ))
```

```
# Adding the output layer
classifier.add(Dense(6, input_dim = 12, activation = 'softmax'))

#%%                    Compiling the ANN

classifier.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

#%%

batch_size= 10
epochs = 100

history = classifier.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(X_test, y_test))

# Evaluate the performance of our trained model
scores = classifier.evaluate(X_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])


#%%              results section

#model_score
history_dict = history.history
train_acc  = history_dict['accuracy']
train_loss = history_dict['loss']
val_acc    = history_dict['val_accuracy']
val_loss   = history_dict['val_loss']

#%%  Plot section

plott_ann(train_acc,val_acc,'Training Accuracy','Validation Accuracy','Accuracy')
plott_ann(train_loss,val_loss,'Training loss','Validation loss','Loss')
plott_ann(val_acc,val_loss,'Validation Accuracy','Valodation Loss','Score')
plott_ann(train_acc,train_loss,'Training Accuracy','Training Loss','Score')

#%%    Model performance

# Part 3 - Making the predictions and evaluating the model

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
```

```
from sklearn.preprocessing import LabelEncoder
y_pred = classifier.predict(X_test)
predictions = np.argmax(y_pred, axis=-1)

label_encoder = LabelEncoder().fit(y_test)
label_y = label_encoder.transform(y_test)

cm = confusion_matrix(label_y, predictions)
print(cm)

from sklearn.metrics import classification_report
print(classification_report(label_y,predictions))



#%% plot ANN configuration

from eiffel2 import builder
# python -m pip install eiffel2

builder([9, 30, 12, 6])
# or the following if you want to have a dark theme
#builder([14, 28, 12, 6], bmode="night")


#%% Plot results Summary

plott_ann(train_acc,val_acc,'Training_Accuracy','Validation_Accuracy','Accuracy')
plott_ann(train_loss,val_loss,'Training_loss','Validation_loss','Loss')
plott_ann(val_acc,val_loss,'Validation Accuracy','Valodation Loss','Score')
plott_ann(train_acc,train_loss,'Training_Accuracy','Training_Loss','Score')

#%% plot all results

import seaborn as sn
import matplotlib.pyplot as plt

sn.set(font_scale=1)
plt.rcParams['figure.figsize']=20,10
sn.set_style("darkgrid")
plt.plot(train_acc, linewidth=4, label  = 'Train Accuracy')
plt.plot(train_loss,linewidth=4, label  = 'Train Loss')
plt.plot(val_acc   ,linewidth=4, label  = 'Validation Accuracy')
plt.plot(val_loss  ,linewidth=4, label  = 'Validation Loss')
plt.ylim(0, 1.05)
```

```
plt.xlabel('epoch',fontsize = 20)
plt.ylabel('Score',fontsize = 20 )
plt.grid(True)
plt.legend()
plt.title("Model Performance", fontsize = 20)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.legend(loc=5, prop={'size': 20})
plt.show()
```