#%%  
import pandas as pd  
import matplotlib.pyplot as plt  
import sklearn  
import numpy as np  
import seaborn as sns

#%%  
from sklearn.model\_selection import KFold, train\_test\_split, cross\_val\_score, StratifiedKFold, cross\_validate, cross\_val\_predict  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.feature\_selection import SelectKBest, f\_classif, chi2  
from sklearn.metrics import confusion\_matrix, classification\_report, log\_loss, hinge\_loss, make\_scorer  
from sklearn.model\_selection import GridSearchCV  
from pandas.plotting import table  
#%%  
data = pd.read\_csv('Machine learning.csv')  
data.head()  
#%% md  
# DATA PROCESSING  
#%%  
data.info()  
#%%  
data.describe()  
#%% md  
# Scaling and Encoding  
#%%  
from sklearn.preprocessing import MinMaxScaler, LabelEncoder  
norm = MinMaxScaler()  
encode = LabelEncoder()  
#%%  
data['Location'] = encode.fit\_transform(data['Location'])  
data.head()  
#%%  
Data\_x = data.iloc[:,:-1]  
Data\_x  
col = list(Data\_x.keys())  
Data\_tf = pd.DataFrame(data=norm.fit\_transform(Data\_x), columns=col)  
Data\_tf['Bioturbation Index'] = data.iloc[:,-1]  
Data\_tf.head()  
#%% md  
# Feature selection using SelectKBest  
#%%  
fe\_selection = SelectKBest(score\_func=f\_classif, k=8).fit\_transform(Data\_tf.iloc[:,:-1],Data\_tf.iloc[:,-1])  
fe\_selection.shape  
feat\_select = pd.DataFrame(fe\_selection)  
feat\_select.head(5)  
#%%  
feat\_select = SelectKBest(score\_func=f\_classif, k=5).fit(data.iloc[:,:-1],data.iloc[:,-1])  
param = pd.DataFrame()  
param['features'] = data.iloc[:,:-1].columns  
param['f\_score'] = feat\_select.scores\_  
param['P\_values'] = feat\_select.pvalues\_  
param['Features\_bool'] = feat\_select.get\_support()  
param= param.sort\_values(by='f\_score', ascending=False)  
param = param.round(5)  
param  
#%%  
plt.figure(figsize=(13,7))  
plt.bar(param['features'], param['P\_values'])  
plt.ylabel('P-Values')  
plt.xticks(rotation = 45)  
plt.title('Features P-values')  
plt.savefig('P-Values')  
#%%  
plt.figure(figsize=(15,8))  
plt.bar(param['features'],1 - param['P\_values'])  
plt.xticks(rotation = 45)  
plt.ylabel('P-Values')  
plt.title('Features {1-} P-values')  
plt.savefig('1-(P-Values)')  
#%%  
New\_data = Data\_tf.drop(['Sample Length','Particle Volume','Dry Weight','Sample Volume','Pore Volume','Bioturbation Index'], axis=1)  
New\_data['Bioturbation Index'] = Data\_tf.iloc[:,-1]  
New\_data.head()  
#%%  
corr = New\_data.corr()  
plt.figure(figsize= (11,8))  
sns.heatmap(corr, annot=True, cbar=True)  
#%%  
#sns.pairplot(New\_data,hue='Bioturbation Index', palette= ['red', 'green','blue','violet','purple'])  
#%% md  
# Data splitting, Cross\_validation and Retraining  
#%%  
X = New\_data.iloc[:,:-1]  
y = New\_data.iloc[:,-1]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y , test\_size=0.2, random\_state=42)  
print('X\_train shape = ', X\_train.shape )  
print('X\_test shape = ', X\_test.shape )  
print('y\_train shape = ', y\_train.shape )  
print('y\_test shape = ', y\_test.shape )  
#%%  
Cl\_svc = SVC()  
Cl\_lda = LinearDiscriminantAnalysis()  
models = [ Cl\_svc, Cl\_lda]  
#%%  
splits = StratifiedKFold(n\_splits=5, shuffle=True)  
SVCcv\_scr = cross\_validate(Cl\_svc, X\_train, y\_train, cv=splits, return\_estimator=True, return\_train\_score=True)  
  
ldacv\_scr = cross\_validate(Cl\_lda, X\_train, y\_train, cv=splits, return\_estimator=True, return\_train\_score=True)  
#%%  
scores = pd.DataFrame({'SVC': (1-SVCcv\_scr['train\_score']),  
 'LDA': (1-ldacv\_scr['train\_score'])})  
scores  
#%%  
print('SVC: ',SVCcv\_scr['train\_score'].mean())  
print('LDA: ',ldacv\_scr['train\_score'].mean())  
#%%  
k\_neighbors = range(3,11,2)  
for k in k\_neighbors:  
 Cl\_knn = KNeighborsClassifier(n\_neighbors=k)  
 knncv\_scr = cross\_validate(Cl\_knn, X\_train, y\_train, cv=splits, return\_estimator=True, return\_train\_score=True)  
 knnscore = pd.DataFrame({k: knncv\_scr['train\_score']})   
 print(knnscore)  
#%% md  
  
#%%  
CRS\_pred = SVCcv\_scr['estimator'][0].predict(X\_train)  
CRS\_report = classification\_report(y\_train,CRS\_pred, target\_names=["Class 0","Class 1","Class 2","Class 3","Class 4","Class 5","Class 6"])  
print(CRS\_report)  
#%%  
CRS\_pred = SVCcv\_scr['estimator'][0].predict(X\_train)  
CRS\_report = confusion\_matrix(y\_train,CRS\_pred)  
print(CRS\_report)  
#%%  
CRK\_pred = knncv\_scr['estimator'][0].predict(X\_train)  
CRK\_report = confusion\_matrix(y\_train,CRK\_pred)  
print(CRK\_report)  
#%%  
CRL\_pred = ldacv\_scr['estimator'][0].predict(X\_train)  
CRL\_report = confusion\_matrix(y\_train,CRL\_pred)  
print(CRL\_report)  
#%% md  
# Optimization with Gridsearchcv  
#%%  
gdModel = KNeighborsClassifier()  
params = [{ 'n\_neighbors':[3,5,7,9],  
 'weights' : ['uniform', 'distance'],  
 'algorithm' : ['auto', 'ball\_tree', 'kd\_tree', 'brute'],  
 'leaf\_size' : [5,10,15,10]  
 }]  
Gdkmodel = GridSearchCV(estimator=gdModel, param\_grid=params, scoring='accuracy', return\_train\_score=True, cv=splits)  
Gdkmodel.fit(X\_train, y\_train)  
print(Gdkmodel.best\_params\_)  
print(Gdkmodel.best\_score\_)  
#%%  
gdsModel = SVC()  
params = [{'C': [0.1, 1, 10, 100, 1000],  
 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],  
 'kernel': ['rbf', 'linear']  
 }]  
Gdsmodel = GridSearchCV(estimator=gdsModel, param\_grid=params, scoring='accuracy', return\_train\_score=True, cv=splits)  
Gdsmodel.fit(X\_train, y\_train)  
print(Gdsmodel.best\_params\_)  
print(Gdsmodel.best\_score\_)  
#%%  
gdlModel = LinearDiscriminantAnalysis()  
grid = dict()  
grid['solver'] = ['svd', 'eigen','lsqr']  
grid['shrinkage'] = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,]  
Gdlmodel = GridSearchCV(gdlModel, param\_grid = grid, scoring='accuracy', return\_train\_score=True, cv=splits)  
Gdlmodel.fit(X\_train, y\_train)  
print('Best Prams:', Gdlmodel.best\_params\_)  
print('Best Score:', Gdlmodel.best\_score\_)  
  
#%%  
from sklearn.metrics import accuracy\_score  
#%%  
for k in range(3,11,2):#'algorithm': 'auto', 'leaf\_size': 5, 'n\_neighbors': 3, 'weights': 'uniform'  
 cl\_knn1 = KNeighborsClassifier(n\_neighbors=k,algorithm='auto',leaf\_size=5, weights='uniform')  
 cl\_knn1.fit(X\_train,y\_train)  
 knn\_pred = cl\_knn1.predict(X\_train)  
 knn\_prob = cl\_knn1.predict\_proba(X\_test)  
 print(cl\_knn1.score(X\_train, y\_train))  
#%%  
C= [1,10,100,1000]  
for c in C:  
 cl\_svm = SVC(C=c, gamma=0.01,kernel='rbf')  
 cl\_svm.fit(X\_train,y\_train)  
 svm\_pred = cl\_svm.predict(X\_train)  
 print(cl\_svm.score(X\_train, y\_train))  
#%% md  
# TRAINING AND TESTING OF BEST HYPERPARAMETERS  
#%%  
#Best Prams: {'shrinkage': 0.9, 'solver': 'eigen'}  
cl\_lda = LinearDiscriminantAnalysis(shrinkage=0.9, solver='eigen')  
cl\_lda.fit(X\_train,y\_train)  
lda\_pred = cl\_lda.predict(X\_train)  
lda\_prob = cl\_lda.predict\_proba(X\_test)  
print(cl\_lda.score(X\_train, y\_train))  
Ldatest\_prediction = cl\_lda.predict(X\_test)  
#%%  
svm\_train = SVC(C=100, kernel='rbf', gamma=1)  
svm\_train.fit(X\_train,y\_train)  
print(svm\_train.score(X\_train,y\_train))  
Svmtest\_prediction = svm\_train.predict(X\_test)  
#%%  
knn\_train = KNeighborsClassifier(n\_neighbors=3)  
knn\_train.fit(X\_train, y\_train)  
print(knn\_train.score(X\_train, y\_train))  
Knntest\_prediction = knn\_train.predict(X\_test)  
#%%  
y\_test = list(y\_test)  
outcomes = pd.DataFrame(data=y\_test, columns=['Actual value'])  
outcomes['KNN\_PRED'] = Knntest\_prediction  
outcomes['LDA\_PRED'] = Ldatest\_prediction  
outcomes['SVM\_PRED'] = Svmtest\_prediction  
outcomes  
#%% md  
  
#%%  
# TESTING OF DATASETS  
print('KNN Accuracy: ', accuracy\_score(y\_test, Knntest\_prediction))  
print('SVM Accuracy: ', accuracy\_score(y\_test, Svmtest\_prediction))  
print('LDA Accuracy: ', accuracy\_score(y\_test, Ldatest\_prediction))  
#%% md  
  
#%% md  
  
#%% md