::: {.cell \_cell\_guid=‘b1076dfc-b9ad-4769-8c92-a6c4dae69d19’ \_uuid=‘8f2839f25d086af736a60e9eeb907d3b93b6e0e5’ execution=‘{“iopub.execute\_input”:“2023-05-18T13:45:05.122442Z”,“iopub.status.busy”:“2023-05-18T13:45:05.122109Z”,“iopub.status.idle”:“2023-05-18T13:45:05.152481Z”,“shell.execute\_reply”:“2023-05-18T13:45:05.151239Z”,“shell.execute\_reply.started”:“2023-05-18T13:45:05.122418Z”}’ trusted=‘true’ execution\_count=1}

import numpy as np # linear algebra  
import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  
import matplotlib.pyplot as plt  
import os  
for dirname, \_, filenames in os.walk('/kaggle/input'):  
 for filename in filenames:  
 print(os.path.join(dirname, filename))

/kaggle/input/transformer/DatasetB.csv  
/kaggle/input/transformer/DatasetA.csv

:::

Here, we have loaded the data and set Furan as the label. At first, we have used 25 percent of the dataset A as the test set to come up with a good model, and then use this model to test in the dataset B.

ds\_A = pd.read\_csv("/kaggle/input/transformer/DatasetA.csv")  
ds\_B = pd.read\_csv("/kaggle/input/transformer/DatasetB.csv")  
  
# Splitting train and test  
from sklearn.model\_selection import train\_test\_split  
train\_set\_A, test\_set\_A = train\_test\_split(ds\_A, test\_size = 0.25, random\_state = 11)  
  
# Setting the labels  
y\_train\_A = train\_set\_A['Furan']  
y\_test\_A = test\_set\_A['Furan']  
  
# Dropping the Furan and Health Index columns  
X\_train\_A = train\_set\_A.drop(["Furan", "HI"], axis = 1)  
X\_test\_A = test\_set\_A.drop(["Furan", "HI"], axis = 1)  
  
# For DatasetB  
y\_B = ds\_B['Furan']  
X\_B = ds\_B.drop(["Furan", "HI"], axis = 1)  
  
# The code below is for the second case, where we train the data for the whole  
# Dataset A and test it on Dataset B  
y\_A = ds\_A['Furan']  
X\_A = ds\_A.drop(["Furan", "HI"], axis = 1)

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5  
 warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

The code below, drops the columns that we don’t need, and only keeps the common features between dataset A and B.

X\_train\_A = X\_train\_A.drop(set(ds\_A.columns) - set(ds\_B.columns), axis=1)  
X\_test\_A = X\_test\_A.drop(set(ds\_A.columns) - set(ds\_B.columns), axis=1)  
X\_A = X\_A.drop(set(ds\_A.columns) - set(ds\_B.columns), axis=1)  
X\_B = X\_B[X\_train\_A.columns]  
X\_train\_A

|  | H2 | Methane | Acetylene | Ethylene | Ethane | Water | Acid | BDV | IFT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 109 | 12.2 | 53.50 | 6.9 | 127.4 | 48.0 | 3 | 0.043 | 83.0 | 20 |
| 566 | 30.2 | 0.00 | 0.0 | 2.6 | 1.1 | 3 | 0.005 | 84.0 | 39 |
| 410 | 45.6 | 18.20 | 0.0 | 1.6 | 1.7 | 5 | 0.005 | 87.0 | 30 |
| 316 | 19.7 | 38.50 | 0.0 | 2.7 | 41.6 | 7 | 0.005 | 50.0 | 32 |
| 678 | 11.0 | 7.60 | 0.0 | 0.3 | 1.6 | 3 | 0.005 | 61.0 | 42 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 269 | 13.7 | 5.10 | 0.0 | 0.4 | 1.1 | 1 | 0.005 | 94.0 | 36 |
| 337 | 32.9 | 3.77 | 0.0 | 0.6 | 2.4 | 6 | 0.005 | 79.0 | 32 |
| 91 | 22.8 | 3.30 | 0.0 | 4.9 | 3.0 | 11 | 0.140 | 88.0 | 16 |
| 80 | 61.2 | 27.30 | 0.0 | 25.6 | 20.8 | 9 | 0.099 | 70.0 | 17 |
| 703 | 58.1 | 9.40 | 0.0 | 1.4 | 1.9 | 5 | 0.005 | 86.0 | 33 |

The code below, discretizes the Furan data into 3 classes.

# define the bin edges for each class  
bins = [-1, 0.1, 1, 100]  
  
# define the labels for each class  
labels = [0, 1, 2]  
  
y\_train\_A = pd.DataFrame(y\_train\_A)  
y\_B = pd.DataFrame(y\_B)  
y\_test\_A = pd.DataFrame(y\_test\_A)  
y\_A = pd.DataFrame(y\_A)  
  
# discretize the data into 3 classes  
y\_train\_A['Class'] = pd.cut(y\_train\_A['Furan'], bins=bins, labels=labels)  
y\_B['Class'] = pd.cut(y\_B['Furan'], bins=bins, labels=labels)  
y\_test\_A['Class'] = pd.cut(y\_test\_A['Furan'], bins=bins, labels=labels)  
y\_A['Class'] = pd.cut(y\_A['Furan'], bins=bins, labels=labels)  
  
y\_train\_A = np.array(y\_train\_A.drop("Furan", axis = 1)).ravel()  
y\_B = np.array(y\_B.drop("Furan", axis = 1)).ravel()  
y\_test\_A = np.array(y\_test\_A.drop("Furan", axis = 1)).ravel()  
y\_A = np.array(y\_A.drop("Furan", axis = 1)).ravel()

The below code is a function to plot the confusion matrix

from sklearn.metrics import confusion\_matrix  
import itertools  
def plot\_confusion\_matrix(cm, classes, normalize=False, cmap=plt.cm.Blues, title='Confusion matrix'):  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print('Confusion matrix, without normalization')  
 print(cm)  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt),  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')

# First case: Training using 75% of the data and testing on the remaining 25%

We have experimented a combination of different models in the ensemble. Although the results were quite similar, we found that a combination of KNN, XGB and logistic regression works best. In the code below we have created a voting classifier consist of these models.

# from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import VotingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from xgboost import XGBClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import AdaBoostClassifier  
  
#log\_clf = LogisticRegression(max\_iter=1000)  
svm\_clf = SVC(probability=True, gamma=0.001)  
knn\_clf = KNeighborsClassifier(n\_neighbors=3)  
xgb\_clf = XGBClassifier(learning\_rate=0.01, n\_estimators=300, max\_depth=3, subsample=0.7)  
mlp\_clf = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=1000)  
nb\_clf = GaussianNB()  
ada\_clf = AdaBoostClassifier(n\_estimators=50, learning\_rate=0.003)  
lr\_clf = LogisticRegression(max\_iter=10000)  
  
voting\_clf = VotingClassifier(  
 estimators=[#('nn', mlp\_clf),  
 #('svc', svm\_clf),  
 ('knn', knn\_clf), #('ada', ada\_clf),('nb', nb\_clf)  
 ('xgb', xgb\_clf), ('lr', lr\_clf)],  
 voting='hard')  
voting\_clf.fit(X\_train\_A, np.array(y\_train\_A).ravel())

VotingClassifier(estimators=[('knn', KNeighborsClassifier(n\_neighbors=3)),  
 ('xgb',  
 XGBClassifier(base\_score=None, booster=None,  
 callbacks=None,  
 colsample\_bylevel=None,  
 colsample\_bynode=None,  
 colsample\_bytree=None,  
 early\_stopping\_rounds=None,  
 enable\_categorical=False,  
 eval\_metric=None,  
 feature\_types=None, gamma=None,  
 gpu\_id=None, grow\_policy=None,  
 importance\_type=None,  
 interaction\_constraints=None,  
 learning\_rate=0.01, max\_bin=None,  
 max\_cat\_threshold=None,  
 max\_cat\_to\_onehot=None,  
 max\_delta\_step=None, max\_depth=3,  
 max\_leaves=None,  
 min\_child\_weight=None, missing=nan,  
 monotone\_constraints=None,  
 n\_estimators=300, n\_jobs=None,  
 num\_parallel\_tree=None,  
 predictor=None, random\_state=None, ...)),  
 ('lr', LogisticRegression(max\_iter=10000))])

Here is a comparison of different models and the voting classifier.

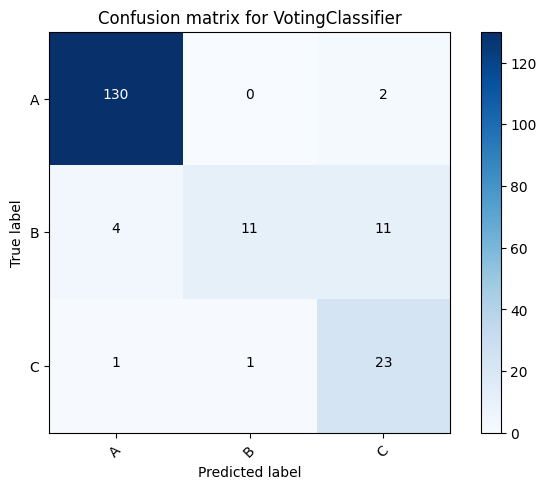
from sklearn.metrics import accuracy\_score  
for clf in (mlp\_clf, svm\_clf, #ada\_clf,  
 knn\_clf, xgb\_clf, #nb\_clf,  
 lr\_clf, voting\_clf):  
 clf.fit(X\_train\_A, y\_train\_A)  
 y\_pred\_A = clf.predict(X\_test\_A)  
 y\_pred\_B = clf.predict(X\_B)  
 print(clf.\_\_class\_\_.\_\_name\_\_ + " for dataset A:", accuracy\_score(y\_test\_A, y\_pred\_A))  
 print(clf.\_\_class\_\_.\_\_name\_\_ + " for dataset B:", accuracy\_score(y\_B, y\_pred\_B))

/opt/conda/lib/python3.10/site-packages/sklearn/neural\_network/\_multilayer\_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.  
 warnings.warn(

MLPClassifier for dataset A: 0.8688524590163934  
MLPClassifier for dataset B: 0.7828746177370031  
SVC for dataset A: 0.8797814207650273  
SVC for dataset B: 0.8103975535168195  
KNeighborsClassifier for dataset A: 0.8360655737704918  
KNeighborsClassifier for dataset B: 0.8042813455657493  
XGBClassifier for dataset A: 0.8961748633879781  
XGBClassifier for dataset B: 0.764525993883792  
LogisticRegression for dataset A: 0.8633879781420765  
LogisticRegression for dataset B: 0.7951070336391437  
VotingClassifier for dataset A: 0.8961748633879781  
VotingClassifier for dataset B: 0.8195718654434251

class\_names = ['A', 'B', 'C']  
voting\_clf.fit(X\_train\_A, y\_train\_A)  
y\_pred\_A = clf.predict(X\_test\_A)  
cnf\_matrix = confusion\_matrix(y\_test\_A, y\_pred\_A)  
np.set\_printoptions(precision=2)  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,  
 title='Confusion matrix for ' + clf.\_\_class\_\_.\_\_name\_\_)  
plt.show()

Confusion matrix, without normalization  
[[130 0 2]  
 [ 4 11 11]  
 [ 1 1 23]]



# Second case: Training using all of the data from Dataset A

So far we have used 75% of Dataset A to train the data and 25% to test it. Here, we used all of the data from Dataset A to train, and then test it on Dataset B.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import VotingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from xgboost import XGBClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import AdaBoostClassifier  
  
lr\_clf = LogisticRegression(max\_iter=10000)  
svm\_clf = SVC(probability=True)  
knn\_clf = KNeighborsClassifier()  
xgb\_clf = XGBClassifier(learning\_rate=0.01, n\_estimators=300, max\_depth=3, subsample=0.7)  
mlp\_clf = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=1000)  
nb\_clf = GaussianNB()  
#ada\_clf = AdaBoostClassifier(n\_estimators=50, learning\_rate=0.003)  
  
voting\_clf = VotingClassifier(  
 estimators=[#('nn', mlp\_clf),  
 #('svc', svm\_clf),  
 ('knn', knn\_clf), #('ada', ada\_clf),('nb', nb\_clf)  
 ('xgb', xgb\_clf), ('lr', lr\_clf)],  
 voting='hard')  
voting\_clf.fit(X\_A, y\_A)

VotingClassifier(estimators=[('knn', KNeighborsClassifier()),  
 ('xgb',  
 XGBClassifier(base\_score=None, booster=None,  
 callbacks=None,  
 colsample\_bylevel=None,  
 colsample\_bynode=None,  
 colsample\_bytree=None,  
 early\_stopping\_rounds=None,  
 enable\_categorical=False,  
 eval\_metric=None,  
 feature\_types=None, gamma=None,  
 gpu\_id=None, grow\_policy=None,  
 importance\_type=None,  
 interaction\_constraints=None,  
 learning\_rate=0.01, max\_bin=None,  
 max\_cat\_threshold=None,  
 max\_cat\_to\_onehot=None,  
 max\_delta\_step=None, max\_depth=3,  
 max\_leaves=None,  
 min\_child\_weight=None, missing=nan,  
 monotone\_constraints=None,  
 n\_estimators=300, n\_jobs=None,  
 num\_parallel\_tree=None,  
 predictor=None, random\_state=None, ...)),  
 ('lr', LogisticRegression(max\_iter=10000))])

from sklearn.metrics import accuracy\_score  
  
for clf in (#mlp\_clf, svm\_clf, #ada\_clf,  
 knn\_clf, xgb\_clf, lr\_clf, #nb\_clf,  
 voting\_clf):  
 clf.fit(X\_A, y\_A)  
 y\_pred\_B = clf.predict(X\_B)  
 print(clf.\_\_class\_\_.\_\_name\_\_ + " for dataset B:", accuracy\_score(y\_B, y\_pred\_B))

KNeighborsClassifier for dataset B: 0.8379204892966361  
XGBClassifier for dataset B: 0.7706422018348624  
LogisticRegression for dataset B: 0.8134556574923547  
VotingClassifier for dataset B: 0.8256880733944955

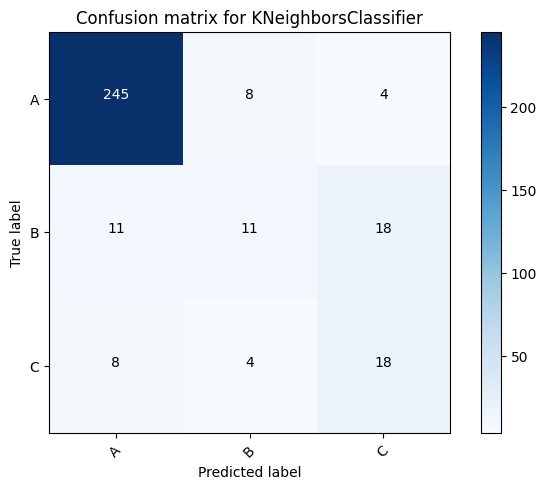
The code below is a function to visualize the confusion matrix

from sklearn.metrics import confusion\_matrix  
import itertools  
def plot\_confusion\_matrix(cm, classes, normalize=False, cmap=plt.cm.Blues, title='Confusion matrix'):  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print('Confusion matrix, without normalization')  
 print(cm)  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt),  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')

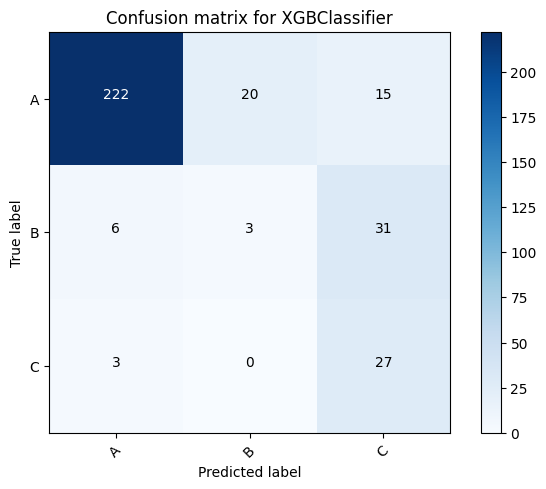
Confusion Matrix:

class\_names = ['A', 'B', 'C']  
  
for clf in (#mlp\_clf, svm\_clf, #ada\_clf,  
 knn\_clf, xgb\_clf, lr\_clf, #nb\_clf,  
 voting\_clf):  
 clf.fit(X\_A, y\_A)  
 y\_pred\_B = clf.predict(X\_B)  
 cnf\_matrix = confusion\_matrix(y\_B, y\_pred\_B)  
 np.set\_printoptions(precision=2)  
 plt.figure()  
 plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,  
 title='Confusion matrix for ' + clf.\_\_class\_\_.\_\_name\_\_)  
 plt.show()

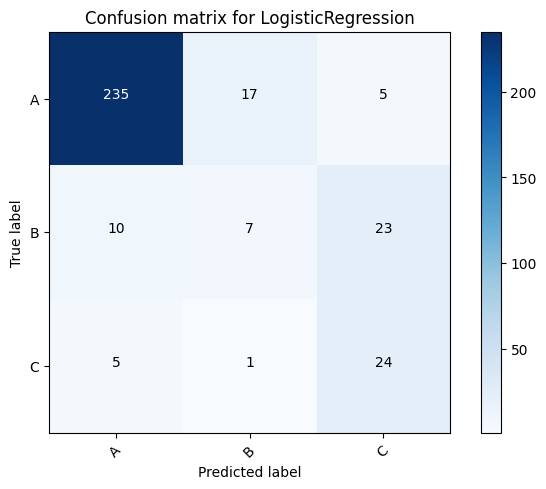
Confusion matrix, without normalization  
[[245 8 4]  
 [ 11 11 18]  
 [ 8 4 18]]



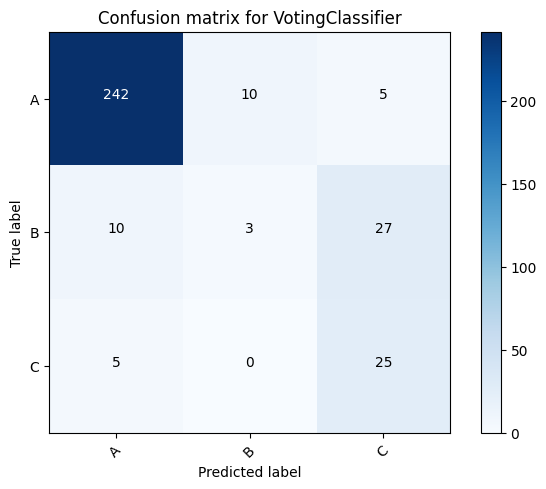
Confusion matrix, without normalization  
[[222 20 15]  
 [ 6 3 31]  
 [ 3 0 27]]



Confusion matrix, without normalization  
[[235 17 5]  
 [ 10 7 23]  
 [ 5 1 24]]



Confusion matrix, without normalization  
[[242 10 5]  
 [ 10 3 27]  
 [ 5 0 25]]



class\_names = ['A', 'B', 'C']  
voting\_clf.fit(X\_A, y\_A)  
y\_pred\_B = voting\_clf.predict(X\_B)  
cnf\_matrix = confusion\_matrix(y\_B, y\_pred\_B)  
np.set\_printoptions(precision=2)  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,  
 title='Confusion matrix for ' + clf.\_\_class\_\_.\_\_name\_\_)  
plt.show()

Confusion matrix, without normalization  
[[242 10 5]  
 [ 10 3 27]  
 [ 5 0 25]]

