experimentdata

May 1, 2021

1 include Libraries

[1]:
```
import os
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LinearRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings

import pickle

warnings.filterwarnings("ignore")
plt.style.use('seaborn-darkgrid')
```

[2]:
```
# display font setting
font = {'family' : 'normal',
```

```
        'weight' : 'bold',
        'size'   : 18}
plt.rc('font', **font)
```

2 Read Data

[3]:
```
xls = pd.ExcelFile('HS-RU-20.xlsx')
NeutralHostile = pd.read_excel(xls, 'NeutralHostile')
HateOffensive = pd.read_excel(xls, 'HateOffensive')
```

3 Preprocess data

[4]:
```
NeutralHostile.shape
```

[4]:
```
(5000, 22)
```

[5]:
```
NeutralHostile.columns
```

[5]:
```
Index(['Sentence', 'label', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4',
       'Unnamed: 5', 'Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8', 'Unnamed: 9',
       'Unnamed: 10', 'Unnamed: 11', 'Unnamed: 12', 'Unnamed: 13',
       'Unnamed: 14', 'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17',
       'Unnamed: 18', 'Unnamed: 19', 'Unnamed: 20', 'Unnamed: 21'],
      dtype='object')
```

[6]:
```
df1 = NeutralHostile.iloc[:,:2].copy()
```

[7]:
```
df1.shape
```

[7]:
```
(5000, 2)
```

[8]:
```
df1['label'] = df1['label'].str.replace('N ','N')
```

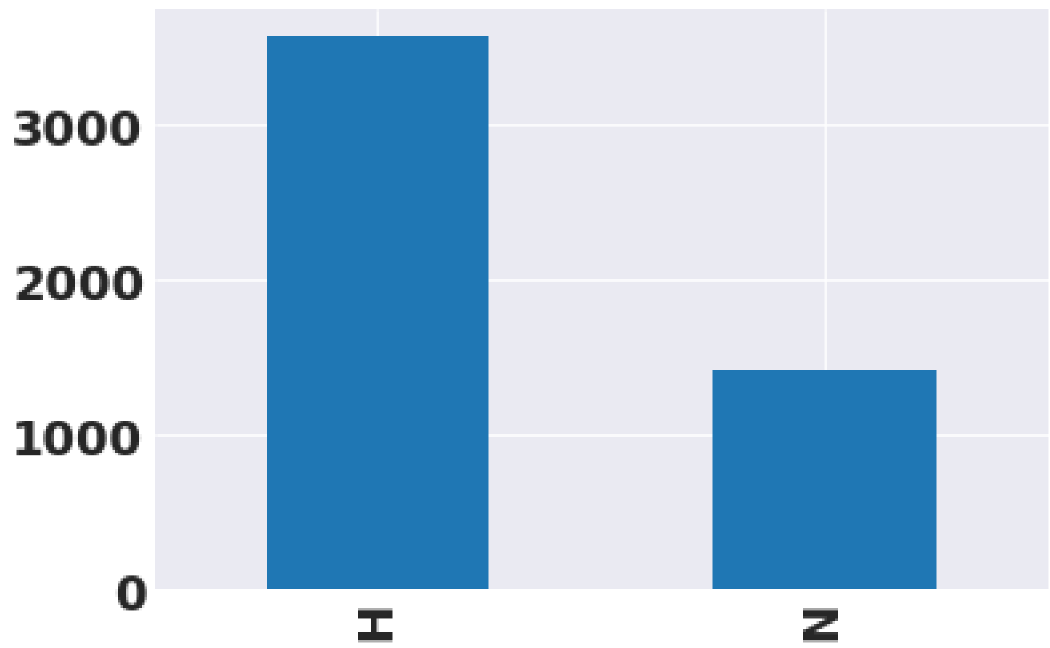
[9]:
```
del NeutralHostile
```

[10]:
```
output = df1.label.value_counts()
```

[11]:
```
output.plot(kind='bar')
```

[11]:
```
<AxesSubplot:>
```

```
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
```



[12]:
```
output
```

[12]:
```
H    3574
N    1426
Name: label, dtype: int64
```

4 Word Unigram

[14]:
```
vectorizer =␣
↪TfidfVectorizer(ngram_range=(2,2),analyzer='word',max_features=1000)
```

[15]:
```
df1values = df1['Sentence'].values
```

[16]:
```
x = vectorizer.fit_transform(df1values)
```

[17]:
```
df1v1 = pd.DataFrame(x.toarray(),columns=vectorizer.get_feature_names())
```

[18]:
```
df1v1.shape
```

[18]:
```
(5000, 1000)
```

5 Concatenate Label with prepared data

[19]:
```
df1final = pd.concat([df1v1,df1['label']],axis=1)
```

[20]:
```
df1final.shape
```

[20]:
```
(5000, 1001)
```

[21]:
```
df1final.to_excel("ProcessedNeutralHostile.xlsx",index=False)
```

6 Label Encodeing

[22]:
```
labelEncoder = LabelEncoder()
```

[23]:
```
df1encoded_final = df1final.copy()
```

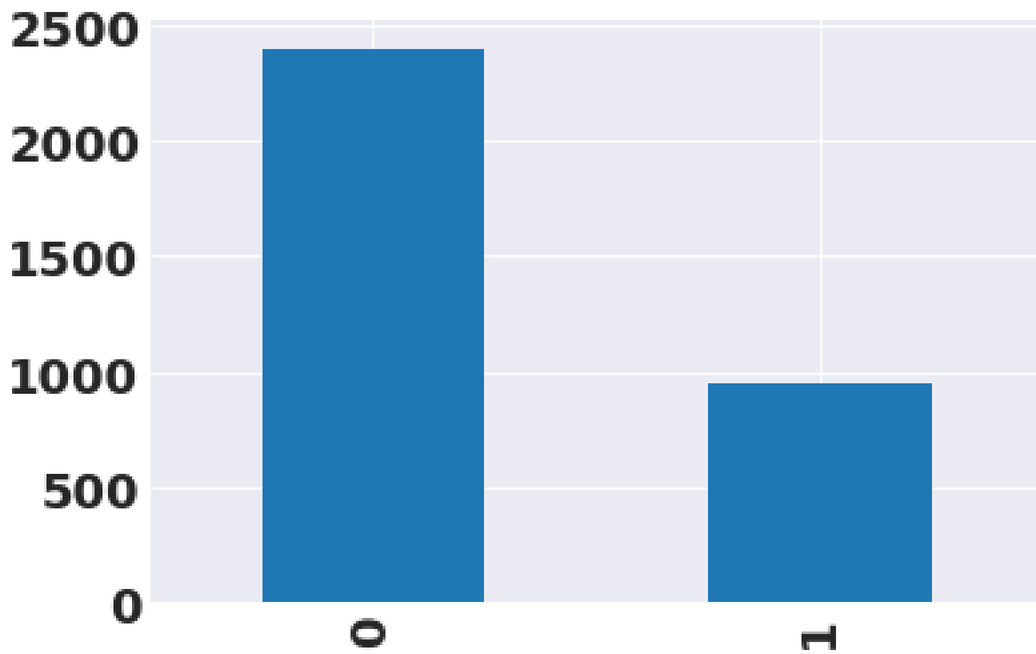
[24]:
```
df1encoded_final['label'] = labelEncoder.fit_transform(df1final['label'])
```

7 Split dataset

[25]:
```
df1encoded_final_train, df1encoded_final_test =␣
 ↪train_test_split(shuffle(df1encoded_final),test_size=.33)
```

[26]:
```
trainclasses = df1encoded_final_train.label.value_counts()
trainclasses.plot(kind='bar')
```

[26]:
```
<AxesSubplot:>
```



[27]:
```
trainclasses
```

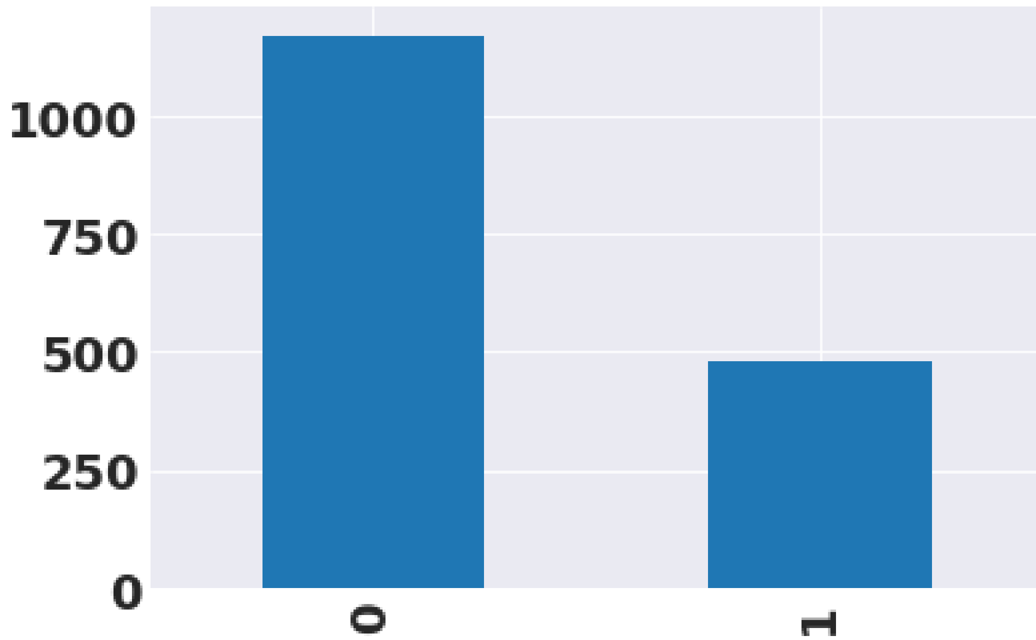
[27]:
```
0    2402
1     948
Name: label, dtype: int64
```

[28]:
```
test_classes = df1encoded_final_test.label.value_counts()
test_classes
```

[28]:
```
0    1172
1     478
Name: label, dtype: int64
```

[29]:
```
test_classes.plot(kind='bar')
```

[29]:
```
<AxesSubplot:>
```



[30]:
```
trainX = df1encoded_final_train.values[:,:-1]
trainY = df1encoded_final_train["label"].values
testX = df1encoded_final_test.values[:,:-1]
testY = df1encoded_final_test["label"].values
```

[31]:
```
train = df1final
```

[32]:
```
train['label'].unique()
```

[32]: array(['H', 'N'], dtype=object)

[ ]:

[33]:
```
# confusion metrix
def print_confusion_matrix(name,confusion_matrix, class_names, figsize = (8,5),␣
↪fontsize=14):
    """Prints a confusion matrix, as returned by sklearn.metrics.
↪confusion_matrix, as a heatmap.

    Arguments
    ---------
    confusion_matrix: numpy.ndarray
        The numpy.ndarray object returned from a call to sklearn.metrics.
↪confusion_matrix.
        Similarly constructed ndarrays can also be used.
```

```
    class_names: list
        An ordered list of class names, in the order they index the given␣
↪confusion matrix.
    figsize: tuple
        A 2-long tuple, the first value determining the horizontal size of the␣
↪ouputted figure,
        the second determining the vertical size. Defaults to (10,7).
    fontsize: int
        Font size for axes labels. Defaults to 14.

    Returns
    -------
    matplotlib.figure.Figure
        The resulting confusion matrix figure
    """
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,␣
↪ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,␣
↪ha='right', fontsize=fontsize)
    plt.ylabel('True label', fontsize=22)
    plt.xlabel('Predicted label', fontsize=22)
    plt.savefig(name+'.jpeg',dpi=600,quality=100,format='jpeg',pad_inches=0.1,
        transparent=True, bbox_inches='tight')
```

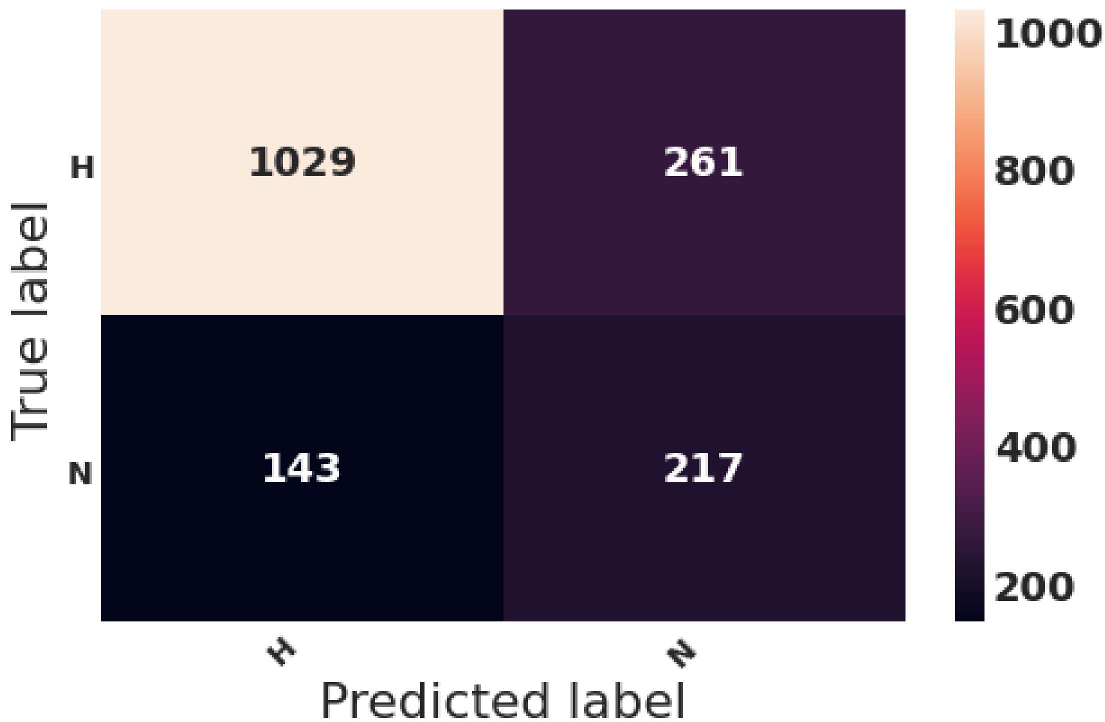
[34]:

```
bernoulliNB =BernoulliNB()
bernoulliNB.fit(trainX,trainY)
BNBScores = cross_val_score(bernoulliNB,trainX ,trainY,␣
↪cv=10,scoring='accuracy')
BNBScore = bernoulliNB.score(testX,testY)
print("Accuracy Score : %f"%(BNBScore))
print("Accuracy Average Score of K Fold : %f"%(BNBScores.mean()))

bnb_perdiction = bernoulliNB.predict(testX)

bnb_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(bnb_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
```

```
#bnb_con_matrix
print_confusion_matrix('BernoulliNB',bnb_con_matrix,train['label'].unique())
```

```
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
```

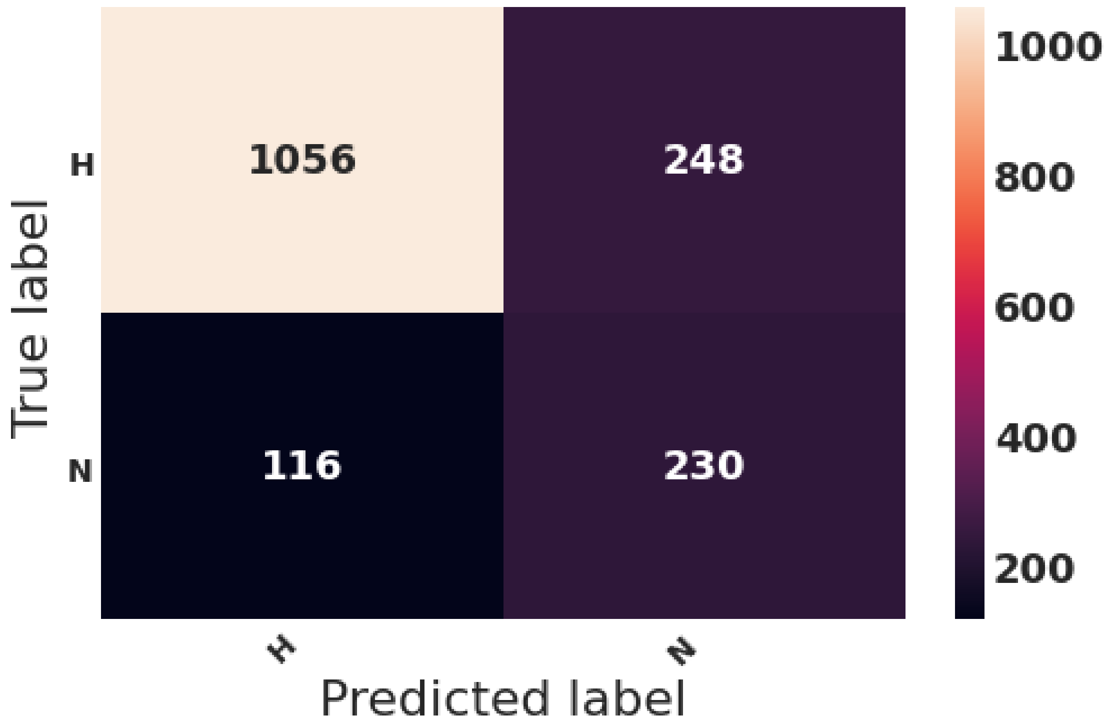
```
Accuracy Score : 0.755152
Accuracy Average Score of K Fold : 0.745373
```



[35]:

```
linearSVC=LinearSVC()
linearSVC.fit(trainX,trainY)
LSVCScores = cross_val_score(linearSVC,trainX ,trainY, cv=10,scoring='accuracy')
LSVCScore = linearSVC.score(testX,testY)
print("Accuracy Score : %f"%(LSVCScore))
print("Accuracy Average Score of K Fold : %f"%(LSVCScores.mean()))

lsvc_perdiction = linearSVC.predict(testX)

lsvc_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(lsvc_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('LinearSVC',lsvc_con_matrix,train['label'].unique())
```

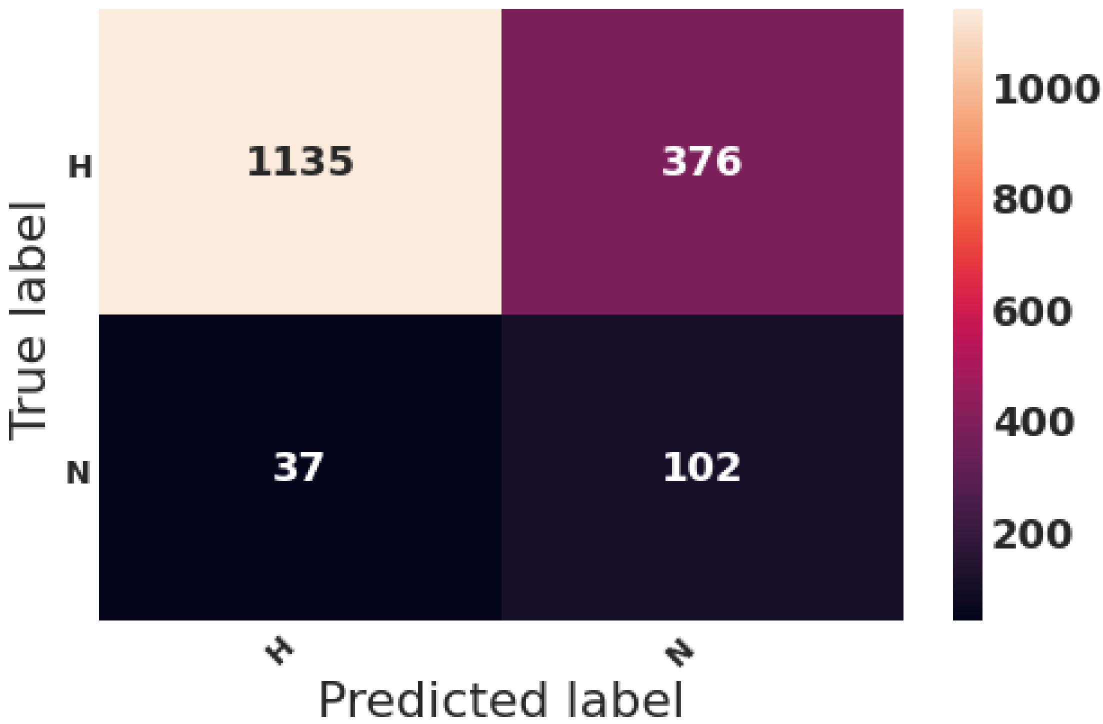
```
Accuracy Score : 0.779394
Accuracy Average Score of K Fold : 0.791940
```



[36]:

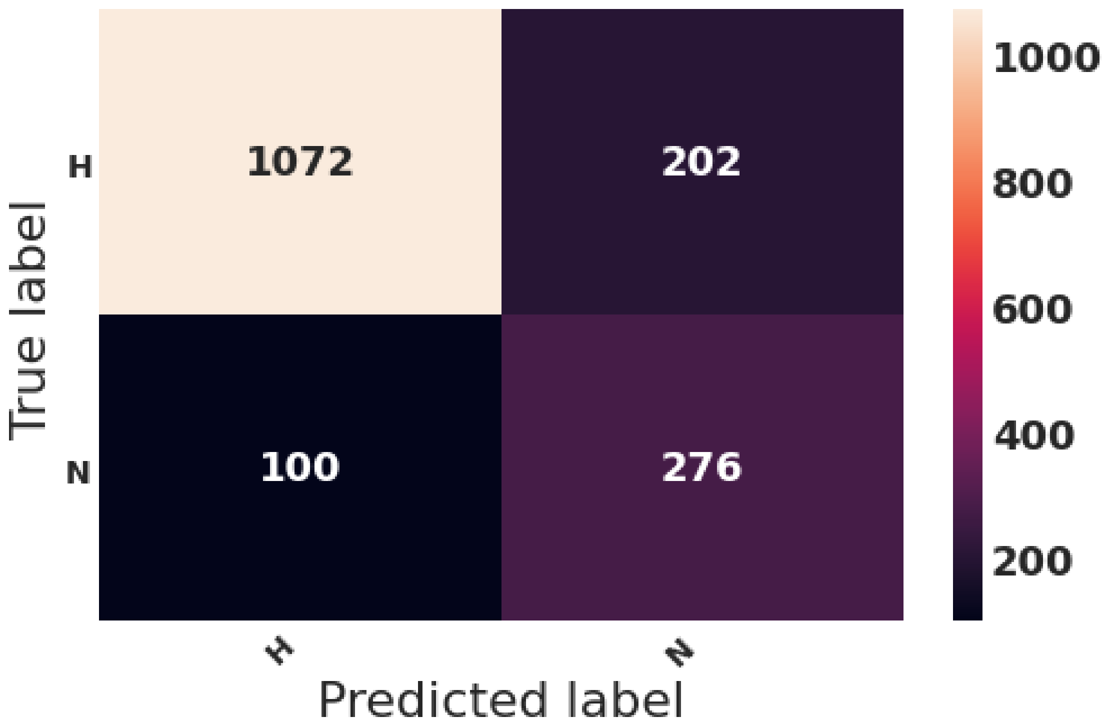
```
logisticRegression =  LogisticRegression();
logisticRegression.fit(trainX,trainY)
LRScores = cross_val_score(logisticRegression,trainX ,trainY,␣
 ↪cv=10,scoring='accuracy')
LRScore = logisticRegression.score(testX,testY)
print("Accuracy Score : %f"%(LRScore))
print("Accuracy Average Score of K Fold : %f"%(LRScores.mean()))

logisticRegression_perdiction = logisticRegression.predict(testX)

LR_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(logisticRegression_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('logisticRegression',LR_con_matrix,train['label'].
 ↪unique())
```

```
Accuracy Score : 0.749697
Accuracy Average Score of K Fold : 0.749851
```



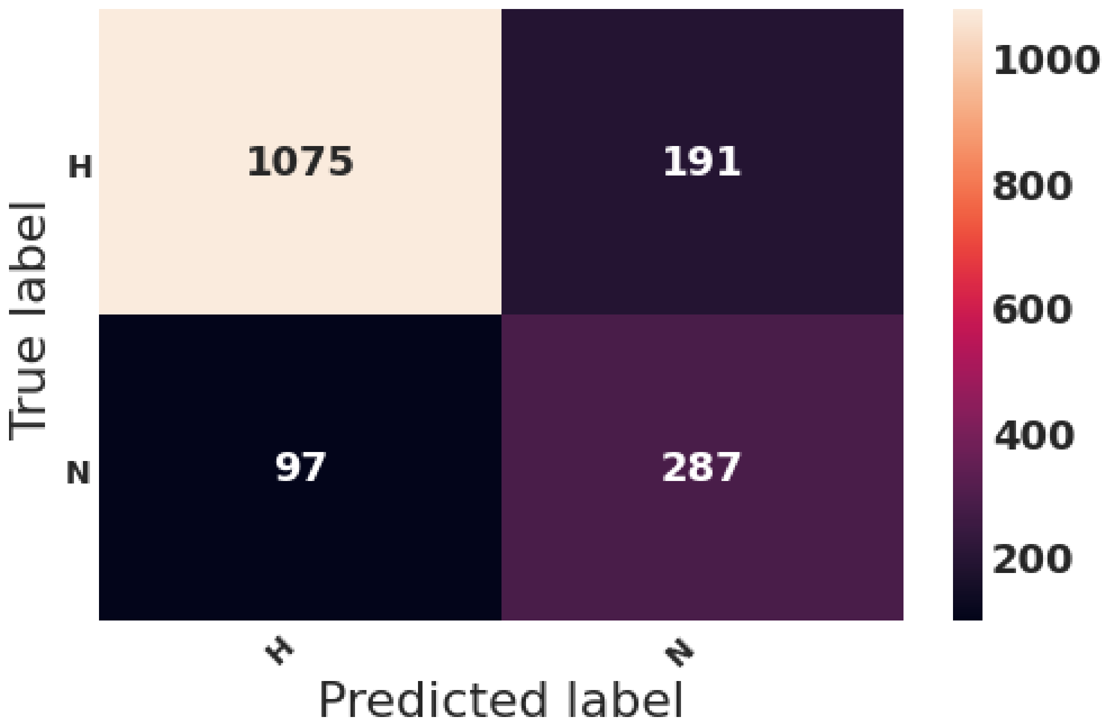
[37]:
```
decisionTreeClassifier = tree.DecisionTreeClassifier()
decisionTreeClassifier.fit(trainX,trainY)
DTCScores = cross_val_score(decisionTreeClassifier,trainX ,trainY,␣
↪cv=10,scoring='accuracy')
DTCScore = decisionTreeClassifier.score(testX,testY)
print("Accuracy Score : %f"%(DTCScore))
print("Accuracy Average Score of K Fold : %f"%(DTCScores.mean()))

DTCS_perdiction = decisionTreeClassifier.predict(testX)

DTC_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(DTCS_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('DecisionTreeClassifier',DTC_con_matrix,train['label'].
↪unique())
```

```
Accuracy Score : 0.816970
Accuracy Average Score of K Fold : 0.814030
```



[38]:
```
kNeighborsClassifier = KNeighborsClassifier(n_neighbors=1)
kNeighborsClassifier.fit(trainX,trainY)
KNCScores = cross_val_score(kNeighborsClassifier,trainX ,trainY,␣
↪cv=10,scoring='accuracy')
KNCScore = kNeighborsClassifier.score(testX,testY)
print("Accuracy Score : %f"%(KNCScore))
print("Accuracy Average Score of K Fold : %f"%(KNCScores.mean()))

knc_perdiction = kNeighborsClassifier.predict(testX)

knc_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(knc_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('KNeighborsClassifier',knc_con_matrix,train['label'].
↪unique())
```

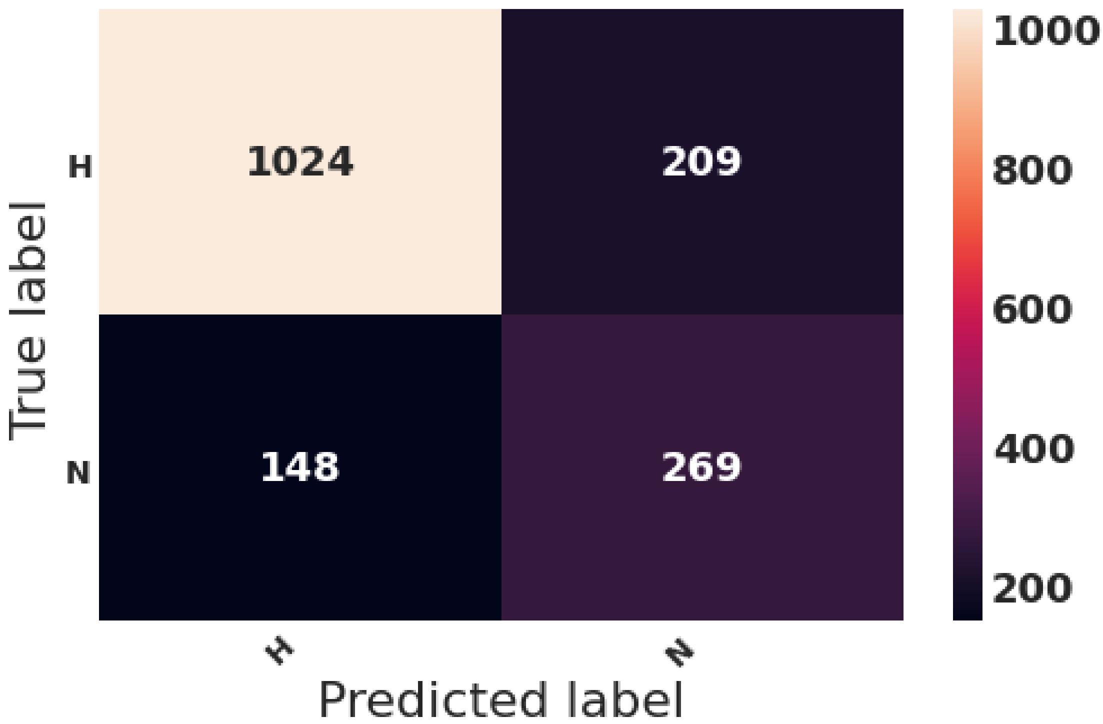
```
Accuracy Score : 0.825455
Accuracy Average Score of K Fold : 0.799701
```



[39]:

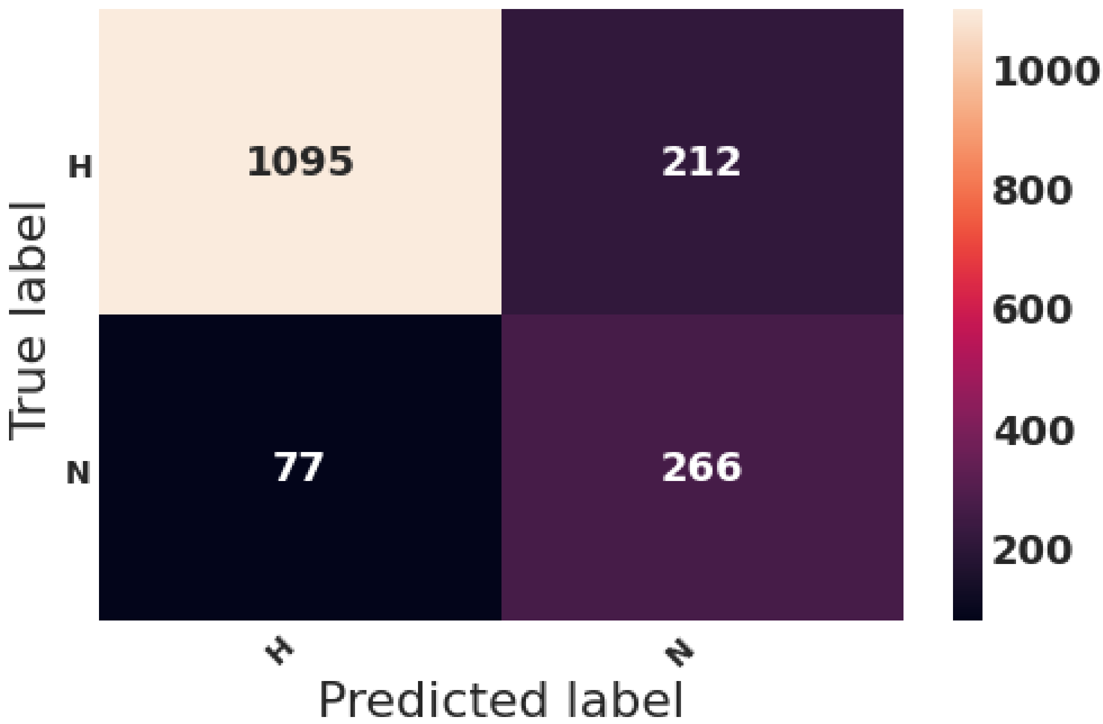
```
lda = LinearDiscriminantAnalysis()
lda.fit(trainX,trainY)
ldascores = cross_val_score(lda,trainX ,trainY, cv=10,scoring='accuracy')
ldascore = lda.score(testX,testY)
print("Accuracy Score : %f"%(ldascore))
print("Accuracy Average Score of K Fold : %f"%(ldascores.mean()))

lda_perdiction = lda.predict(testX)

lda_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(lda_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('LinearDiscriminantAnalysis',lda_con_matrix,train['label'].
 ↪unique())
```

```
Accuracy Score : 0.783636
Accuracy Average Score of K Fold : 0.765373
```



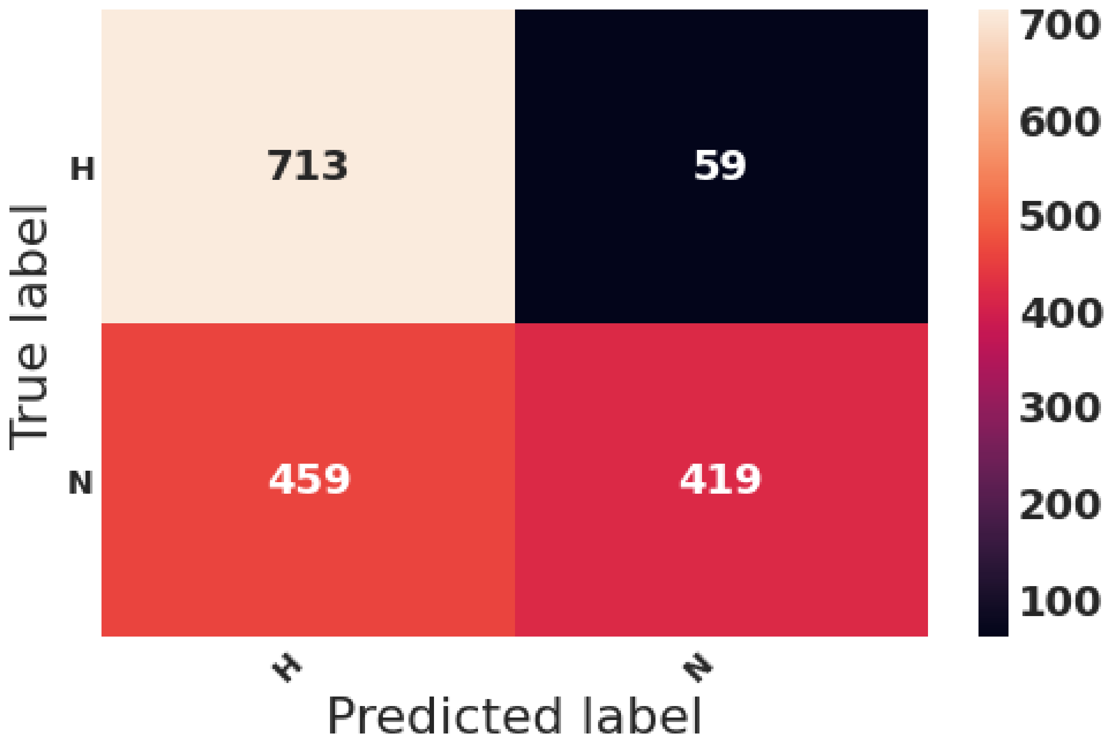
[40]:
```
randomForestClassifier = RandomForestClassifier()
randomForestClassifier.fit(trainX,trainY)
RFCScores = cross_val_score(randomForestClassifier,trainX ,trainY,␣
↪cv=10,scoring='accuracy')
RFCScore = randomForestClassifier.score(testX,testY)
print("Accuracy Score : %f"%(RFCScore))
print("Accuracy Average Score of K Fold : %f"%(RFCScores.mean()))

rfc_perdiction = randomForestClassifier.predict(testX)

rfc_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(rfc_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('RandomForestClassifier',rfc_con_matrix,train['label'].
↪unique())
```

```
Accuracy Score : 0.824848
Accuracy Average Score of K Fold : 0.822985
```



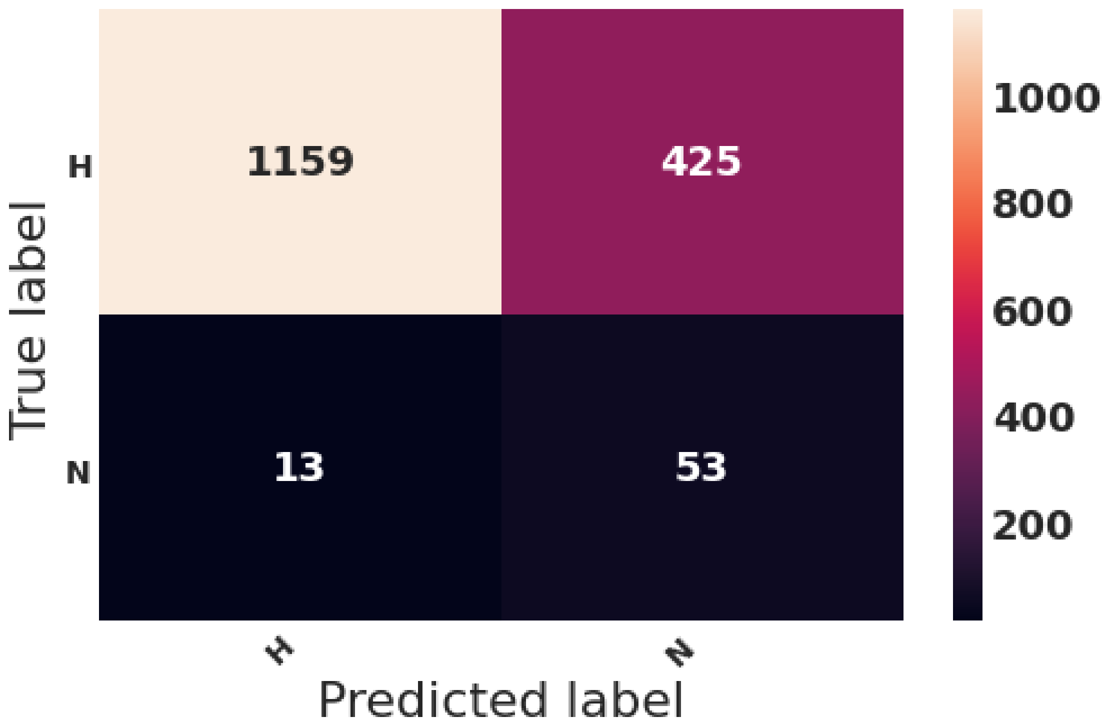
[41]:
```
gaussianNB = GaussianNB()
gaussianNB.fit(trainX,trainY)
GNBScores = cross_val_score(gaussianNB,trainX ,trainY, cv=10,scoring='accuracy')
GNBScore = gaussianNB.score(testX,testY)
print("Accuracy Score : %f"%(GNBScore))
print("Accuracy Average Score of K Fold : %f"%(GNBScores.mean()))

GNB_perdiction = gaussianNB.predict(testX)

GNB_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(GNB_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('gaussianNB',GNB_con_matrix,train['label'].unique())
```

```
Accuracy Score : 0.686061
Accuracy Average Score of K Fold : 0.684776
```



[42]:
```
GBC = GradientBoostingClassifier()
GBC.fit(trainX,trainY)
GBCScores = cross_val_score(GBC,trainX ,trainY, cv=10,scoring='accuracy')
GBCScore = GBC.score(testX,testY)
print("Accuracy Score : %f"%(GBCScore))
print("Accuracy Average Score of K Fold : %f"%(GBCScores.mean()))

GBC_perdiction = GBC.predict(testX)

GBC_con_matrix = confusion_matrix(
    labelEncoder.inverse_transform(GBC_perdiction),
    labelEncoder.inverse_transform(testY),
    train['label'].unique()
)
print_confusion_matrix('GradientBoostingClassifier',GBC_con_matrix,train['label'].
 ↪unique())
```

```
Accuracy Score : 0.734545
Accuracy Average Score of K Fold : 0.736716
```



[43]:
```
predictionModals = {
    'Accuracy': [LRScore, RFCScore, GBCScore,␣
 ↪BNBScore,DTCScore,KNCScore,ldascore,GNBScore,LSVCScore],
    'Model': ['LR', 'RFC', 'GBC', 'BerNB','DTC',"KNC",'LinDA','GausNB','LinSVC']

}
prodictionModelDF = pd.DataFrame(predictionModals, columns=['Accuracy',␣
 ↪'Model'])
print("detail performance of all parameters : ")
prodictionModelDF
```

```
detail performance of all parameters :
```

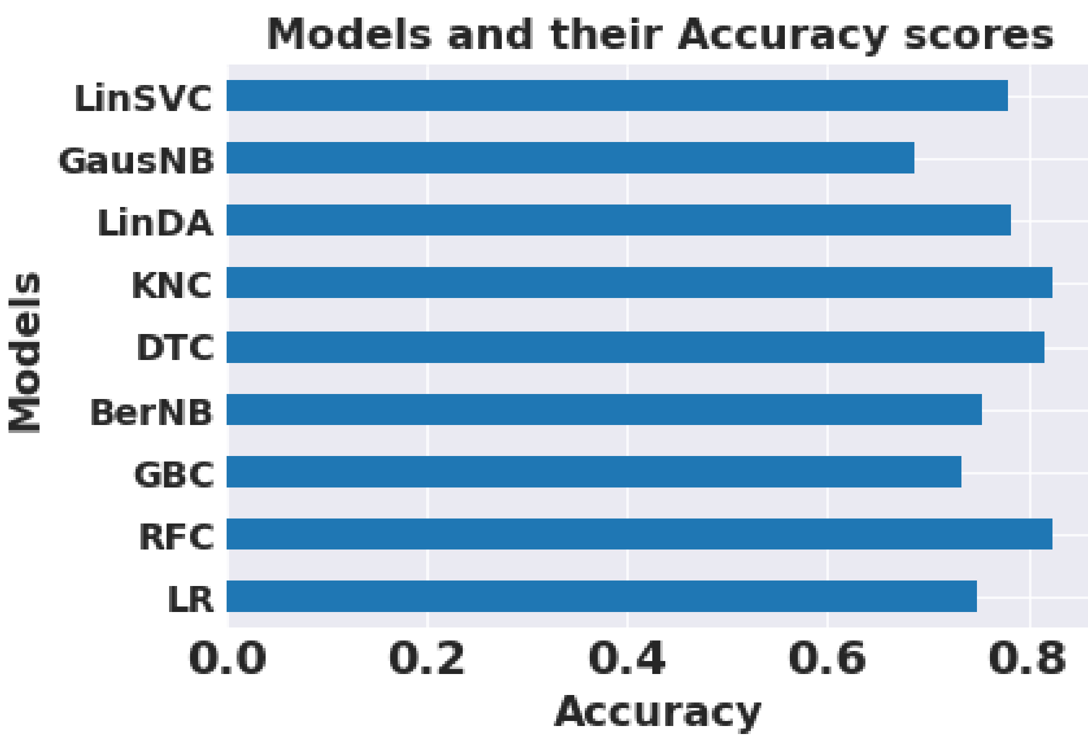
[43]:

	Accuracy	Model
0	0.749697	LR
1	0.824848	RFC
2	0.734545	GBC
3	0.755152	BerNB
4	0.816970	DTC
5	0.825455	KNC
6	0.783636	LinDA
7	0.686061	GausNB
8	0.779394	LinSVC

[44]:

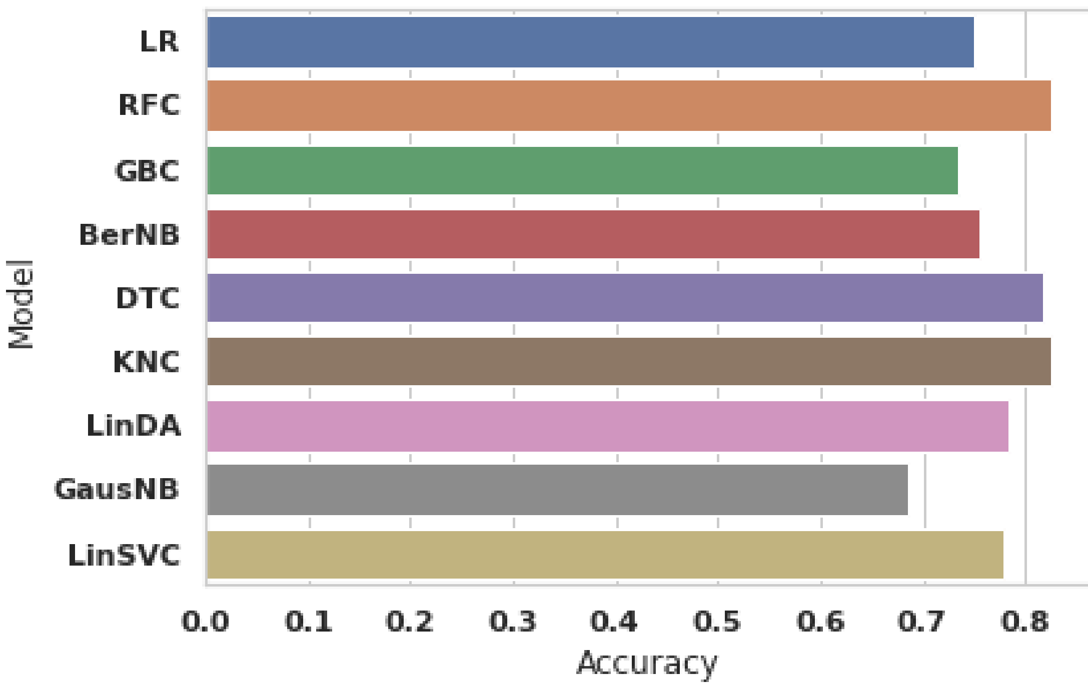
```
plt.rc('ytick', labelsize=14)
Plot = prodictionModelDF.plot.barh(x='Model', y='Accuracy', legend=None)
Plot.set_ylabel('Models',fontdict={'fontsize': 16, 'fontweight': 'heavy'})
Plot.set_xlabel('Accuracy',fontdict={'fontsize': 16, 'fontweight': 'heavy'})
Plot.set_title("Models and their Accuracy scores",fontdict={'fontsize': 16,␣
↪'fontweight': 'heavy'})
fig = Plot.get_figure()
fig.savefig("ModelsAccuracy.
↪jpeg",dpi=600,format='jpeg',quality=100,pad_inches=0.1,
        transparent=True, bbox_inches='tight')
```

```
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
```

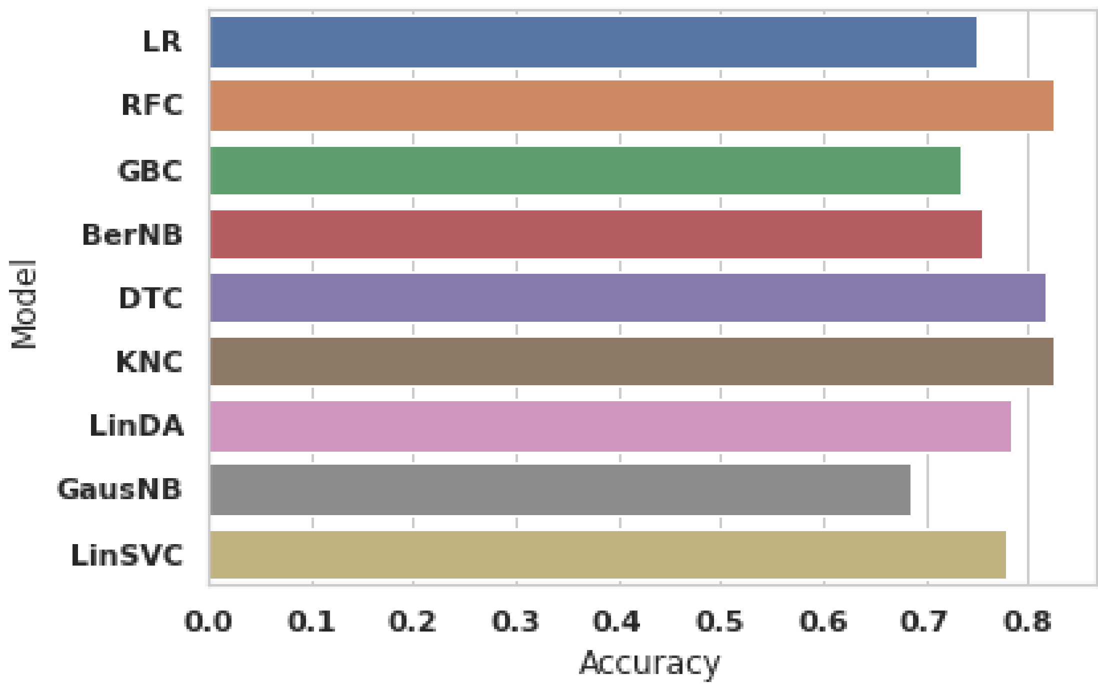


[51]:

```
```



[54]:
```
sns.set_theme(style="whitegrid")
ax = sns.barplot(x="Accuracy", y="Model", data=prodictionModelDF)
fig = ax.get_figure()
fig.savefig("ModelsAccuracy_seaborn.
 ↪jpeg",dpi=600,format='jpeg',quality=100,pad_inches=0.1,
        transparent=True, bbox_inches='tight')
```


LR
RFC
GBC
BerNB
DTC
KNC
LinDA
GausNB
LinSVC
Model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Accuracy

```
[ ]:
```