Gas Sensor Array Drift Estimation at Different Concentrations

Vinayak Gupta

Aim of the project

The paper presents an analysis of the application of various supervised learning algorithms on the "Gas Sensor Array Drift Dataset at Various Concentrations" of UCI Machine Learning repository. The effectiveness of an algorithm depends a lot on the kind of the problem it is being applied on and the results of the these multiple classification algorithms justify that. The project aims at carrying out drift analysis of the given dataset. Given the features, one has to determine to which of the 6 classes the group of features belong. Based on the results of the algorithms, we can see that the problem is highly linearly separable and hence, classification techniques like SVM, Neural Networks, Logistic Regression perform really well. The results also prove the ineffeciency of other algorithms like Naïve Bayes, Random Forest, KNN.

Dataset

The data set has 13,910 measurements which need to be classified into 6 gases. The data was collected between January 2008 to February 2011 in a gas delivery platform facility situated at the ChemoSignals Laboratory in UCSD. The 6 gases are Ammonia, Acetaldehyde, Acetone, Ethylene, Ethanol, and Toluene.

The dataset is organized into ten batches as shown in the Figure 1 and Figure 2. To ensure uniformity, the data was distributed in such a way.

Batch ID	Month IDs
Batch 1	Months 1 and 2
Batch 2	Months 3, 4, 8, 9 and 10
Batch 3	Months 11, 12, and 13
Batch 4	Months 14 and 15
Batch 5	Month 16
Batch 6	Months 17, 18, 19, and 20
Batch 7	Month 21
Batch 8	Months 22 and 23
Batch 9	Months 24 and 30
Batch 10	Month 36

Figure 1: Batch Details

Batch ID: Ethanol, Ethylene, Ammonia, Acetaldehyde, Acetone, Toluene
Batch 1: 83, 30, 70, 98, 90, 74
Batch 2: 100, 109, 532, 334, 164, 5
Batch 3: 216, 240, 275, 490, 365, 0
Batch 4: 12, 30, 12, 43, 64, 0
Batch 5: 20, 46, 63, 40, 28, 0
Batch 6: 110, 29, 606, 574, 514, 467
Batch 7: 360, 744, 630, 662, 649, 568
Batch 8: 40, 33, 143, 30, 30, 18
Batch 9: 100, 75, 78, 55, 61, 101
Batch 10: 600, 600, 600, 600, 600, 600

Figure 2: Batch measurements

Each row starts with a numerical value between 1 and 6 representing the following:

1: Ethanol; 2: Ethylene; 3: Ammonia; 4: Acetaldehyde; 5: Acetone; 6: Toluene

The data format is of x:v, where x stands for feature and v stands for value.

Data Preprocessing

Python libraries like numpy, pandas were used for preprocessing the data. As the data was present in libsvm format as given above, a combination of for loops, split function, append function ad dictionary were used to transfer the data into a dataframe. The dataframe created contained 130 columns which included the 128 features, a column for the concetration and a column for the target i.e the class.

Also, all the missing data was replaced by the mean of the entire column using the Scikit-learn library of python. The data was scaled as well so that any feature does not individually determine the classes while clustering. Moreover, the data was apportioned into a 80-20 training-test split using the train_test_split module from the Scikit-learn library. The code given in the following figures are used for preprocessing the data.

```
f=open('batch1.dat','r')
df = pd.DataFrame()
for line in f:
    l1 = []
    l2 = []
    l1 = line.split(" ")
    l1 = l1[:-1]
    for ele in l1:
        l2.append(ele.split(":"))
    ele1 = l2[0]
    ele2 = ["Target",int(ele1[0].split(";")[0])]
    ele3 = ["Concentration",float(ele1[0].split(";")[1])]
    l2[0] = ele2
    l2.append(ele3)
    d1 = dict()
    for ele in l2:
        d1[ele[0]] = [ele[1]]
    df_temp = pd.DataFrame.from_dict(d1)
    df = df.append(df_temp)

print (df.shape)
df.head()
```

```
(445, 130)
```

Figure 3: Splitting data in batch file and adding it to dataframe

```
from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values='NaN', strategy ='mean', axis=0)
imputer.fit(X[:,:])
X[:,]= imputer.transform(X[:,:])
X
```

Figure 4: Replacing the Nan values by mean of the column

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train= sc_X.fit_transform(X_train)
X_test= sc_X.transform(X_test)
```

Figure 5: Apportioning and scaling the data

Classification Algorithms

A total of 10 classification algorithms were tested on the dataset. First, the algorithms were applied without taking the concentration feature into account. Those algorithms which provided poor results in such case were filtered out and the remaining algorithms were then implemented while taking concentration as well. The 10 algorithms implemented are:

1) Logistic regression
2) KNN with K= 5
3) KNN with K= 9
4) Linear SVM
5) Kernel SVM
6) Decision Tree
7) Random Forest (n=10)
8) Random Forest (n=50)
9) Artificial Neural Networks with 2 hidden layers
10) Naïve Bayes

The pros and cons of all the classification algorithms is given in the following figure 6:

Classification Model	Pros	Cons
Logistic Regression	Probabilistic approach, gives informations about statistical significance of features	The Logistic Regression Assumptions
K-NN	Simple to understand, fast and efficient	Need to choose the number of neighbours k
SVM	Performant, not biased by outliers, not sensitive to overfitting	Not appropriate for non linear problems, not the best choice for large number of features
Kernel SVM	High performance on nonlinear problems, not biased by outliers, not sensitive to overfitting	Not the best choice for large number of features, more complex
Naive Bayes	Efficient, not biased by outliers, works on nonlinear problems, probabilistic approach	Based on the assumption that features have same statistical relevance
Decision Tree Classification	Interpretability, no need for feature scaling, works on both linear / nonlinear problems	Poor results on too small datasets, overfitting can easily occur
Random Forest Classification	Powerful and accurate, good performance on many problems, including non linear	No interpretability, overfitting can easily occur, need to choose the number of trees

Figure 6: Pros and cons of supervised learning algorithms

Confusion Matrix

A confusion matrix (Kohavi and Provost, 1998) contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix

		Predicted	
		Negative	Positive
Actual	Negative	**a**	**b**
	Positive	**c**	**d**

Figure 7: Confusion Matrix

The accuracy of algorithms was defined by the following formula:

$$Accuracy(A) = \frac{a + d}{\mathrm{a} + \mathrm{b} + \mathrm{c} + \mathrm{d}}$$

The Scikit-learn library of python provides for the implementation of confusion matrix. The following code is used for the same:

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,new_pred)
cm
```

```
array([[17,  0,  0,  0,  0,  0],
       [ 0, 23,  0,  0,  0,  0],
       [ 1,  0, 12,  0,  0,  0],
       [ 0,  0,  0,  6,  0,  0],
       [ 0,  0,  0,  0, 14,  0],
       [ 0,  0,  0,  0,  0, 16]], dtype=int64)
```

Figure 8: Confusion matrix in Python

Test Set

20% of data of every batch comprises of the test set. The data is randomly selected.

Batch	Test Size (20%)
1	89
2	249
3	318
4	33
5	40
6	460
7	723
8	59
9	94
10	720

Results

As mentioned before, the results are calculated in 2 steps. First, we test the algorithms without taking concentration into account. The following figure gives the number of incorrect predictions for each algorithm:

Batch	Test Size	Log Reg.	KNN N=5	KNN N=9	SVM	Kernel SVM	Decision Tree	Random Forest (n=10)	Random Forest (n=50)	Neural Networks	Naïve Bayes
1	89	4	3	7	3	8	6	4	2	2	30
2	249	2	1	2	1	1	3	0	0	0	15
3	318	4	3	5	1	5	1	1	2	2	20
4	33	0	0	2	0	0	0	0	0	0	1
5	40	1	1	1	1	1	3	2	1	0	1
6	460	4	5	6	3	5	6	9	5	4	84
7	723	2	3	3	2	3	14	2	2	2	231
8	59	3	1	3	2	3	3	1	1	2	3
9	94	0	0	0	0	0	0	0	0	0	3
10	720	4	15	21	4	34	25	14	11	5	303
	2785	24	32	50	17	60	61	33	24	17	691
	Accuracy	99.14	98.85	98.20	99.39	97.85	97.81	98.82	99.14	99.39	75.19

Table 1. Results without taking concentration into account

Batch	Test Size	Log Reg.	KNN N=5	SVM	Random Forest (n=50)	Neural Networks
1	89	3	3	3	6	1
2	249	1	1	1	0	0
3	318	3	3	1	2	0
4	33	0	0	0	0	0
5	40	1	1	1	0	0
6	460	3	4	3	5	4
7	723	2	3	2	2	2
8	59	3	1	2	2	2
9	94	0	0	0	0	0
10	720	2	15	4	10	5
	2785	19	31	17	27	14
	Accuracy	99.32	98.89	99.39	99.03	99.50

Table 2. Results with concentration feature

As we can see, algorithms that work good on linear separable data perform best. Naïve Bayes performs very poorly as it is good with non linear problems. The good and neutral algorithms are further selected for the 2nd phase of testing involving the concentration feature as well.

- The best accuracy was shown by neural networks having a total of 14 incorrect predictions on a test set of 2785. (99.5%)
- This was followed by SVM which had a total of 17 incorrect predictions. (Accuracy = 99.39%)
- Logistic regression also performed good with an accuracy of 99.31 %
- Worst classification accuracy was by Naïve Bayes algorithm with just a 75.18% accuracy.

Conclusion

The problem was linearly separable, therefore Kernel SVM, Naïve Bayes and KNN had comparatively poor results while SVM and Logistic Regression gave good results. Neural Networks is efficient in every case and therefore gave good results. Decision Tree is poor on small datasets, therefore was not effective. And since, decision tree was not effective Random Forest also did not perform much well.

References

1) A Vergara, S Vembu, T Ayhan, M Ryan, M Homer, R Huerta. "Chemical gas sensor drift compensation using classifier ensembles." Sensors and Actuators B: Chemical 166 (2012): 320-329.

2) I Rodriguez-Lujan, J Fonollosa, A Vergara, M Homer, R Huerta. "On the calibration of sensor arrays for pattern recognition using the minimal number of experiments." Chemometrics and Intelligent Laboratory Systems 130 (2014): 123-134.

3) Sathishkumar, E & Kuttiyannan, Dr. Thangavel & Arul, D & Daniel, Arul. (2013). Effective Clustering Algorithm for Gas Sensor Array Drift Dataset. 3.

Annexure

Code for Neural Networks taking concetration into account on batch file 1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

f=open('batch1.dat','r')
df = pd.DataFrame()
for line in f:
    l1 = []
    l2 = []
    l1 = line.split(" ")
    l1 = l1[:-1]
    for ele in l1:
        l2.append(ele.split(":"))
    ele1 = l2[0]
    ele2 = ["Target",int(ele1[0].split(";")[0])]
    ele3 = ["Concentration",float(ele1[0].split(";")[1])]
    l2[0] = ele2
    l2.append(ele3)
    d1 = dict()
    for ele in l2:
        d1[ele[0]] = [ele[1]]
    df_temp = pd.DataFrame.from_dict(d1)
    df = df.append(df_temp)

print (df.shape)
df.head()

X= df.iloc[:,1:].values
y= df.iloc[:,0].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_y_1 = LabelEncoder()
y = labelencoder_y_1.fit_transform(y)

from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values='NaN', strategy ='mean', axis=0)
imputer.fit(X[:,:])
X[:,]= imputer.transform(X[:,:])
X
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0)

from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train= sc_X.fit_transform(X_train)
X_test= sc_X.transform(X_test)

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

classifier= Sequential()
classifier.add(Dense(output_dim=65,init='uniform',activation='relu',input_dim=129))

classifier.add(Dense(output_dim=65,init='uniform',activation='relu'))
classifier.add(Dense(output_dim=6,init='uniform',activation='softmax'))
classifier.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

classifier.fit(X_train,y_train,batch_size=10,nb_epoch=100)

y_pred=classifier.predict(X_test)
new_pred=[]
for ele in y_pred:
    index_max = np.argmax(ele)
    new_pred.append(index_max)

from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,new_pred)
cm

from sklearn.metrics import accuracy_score

Accuracy_Score = accuracy_score(y_test, new_pred)

print(Accuracy_Score.mean()*100)
```

Code for SVM taking concetration into account on batch file 10

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

f=open('batch10.dat','r')
df = pd.DataFrame()
for line in f:
    l1 = []
    l2 = []
    l1 = line.split(" ")
    l1 = l1[:-1]
    for ele in l1:
        l2.append(ele.split(":"))
    ele1 = l2[0]
    ele2 = ["Target",int(ele1[0].split(";")[0])]
    ele3 = ["Concentration",float(ele1[0].split(";")[1])]
    l2[0] = ele2
    l2.append(ele3)
    d1 = dict()
    for ele in l2:
        d1[ele[0]] = [ele[1]]
    df_temp = pd.DataFrame.from_dict(d1)
    df = df.append(df_temp)

print (df.shape)
df.head()

X= df.iloc[:,0:129].values
y= df.iloc[:,129].values

from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values='NaN', strategy ='mean', axis=0)
imputer.fit(X[:,:])
X[:,]= imputer.transform(X[:,:])
X

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0)

from sklearn.preprocessing import StandardScaler
sc_X= StandardScaler()
X_train= sc_X.fit_transform(X_train)
X_test= sc_X.transform(X_test)
```

```
from sklearn.svm import SVC
classifier= SVC(kernel='linear',random_state=0)
classifier.fit(X_train,y_train)

y_pred=classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,y_pred)
cm

from sklearn.metrics import accuracy_score
Accuracy_Score = accuracy_score(y_test, y_pred)
print(Accuracy_Score.mean()*100)
```