Elective Course
Course Code: CS4103
Autumn 2025-26

Lecture #50

Artificial Intelligence for Data Science

Week-14:

MACHINE LEARNING (Part XVIII)

Exploring Machine Learning Models for Acute Aquatic Toxicity Prediction

Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computational and Data Sciences
Indian Institute of Science Education and Research Kolkata, India 741246

Acute Aquatic Toxicity Prediction Dataset

2

- Dataset containing values for 8 attributes (molecular descriptors) of 546 chemicals used to predict quantitative acute aquatic toxicity towards Daphnia Magna.
  - 8 molecular descriptors:
    - TPSA(Tot) (Molecular properties),
    - SAacc (Molecular properties),
    - H-050 (Atom-centred fragments),
    - MLOGP (Molecular properties),
    - RDCHI (Connectivity indices),
    - GATS1p (2D autocorrelations),
    - nN (Constitutional indices),
    - C-040 (Atom-centred fragments).
  - Target (Toxicity Level):
    - High (2)
    - Mild (1)
    - Low (0)

Acute Aquatic Toxicity Prediction: Dataset Loading

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

dataset=pd.read_excel("F:/CS4103/code/Aquatic_Toxicity.xlsx")
print(dataset.info())
print(dataset.head())
```

	F1	F2	F3	F4	F5	F6	F7	F8	Target
0	0.00	0.0	0	2.419	1.225	0.667	0	0	1
1	0.00	0.0	0	2.638	1.401	0.632	0	0	2
2	9.23	11.0	0	5.799	2.930	0.486	0	0	2
3	9.23	11.0	0	5.453	2.887	0.495	0	0	2
4	9.23	11.0	0	4.068	2.758	0.695	0	0	2

Acute Aquatic Toxicity Prediction: Exploring Dataset

```
#Dataset column names
print("\nName of the columns in the dataset:")
print(list(dataset.columns))
input_features = list(dataset.columns[:-1])
class_col=dataset.columns[-1]
print("Name of the input feature
columns:",input_features)
print("Name of the class/label column:",class_col)
#Number of rows or instances and number of columns
features
print("\nTotal number of data
points/examples/instances:", dataset.shape[0])
print("Total number of input features:",
dataset.shape[1]-1)
#Dataset description
print("\nDescription of the dataset:")
print(dataset[dataset.columns[:-1]].describe())
```

```
Name of the columns in the dataset:
['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7',
'F8', 'Target']
Name of the input feature columns: ['F1',
'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8']
Name of the class/label column: Target
```

```
Total number of data points/examples/
instances: 546
Total number of input features: 8
```

Description of the dataset:

	F1	F2	F3	...	F6	F7	F8
count	546.000000	546.000000	546.000000	...	546.000000	546.000000	546.000000
mean	48.472930	58.869018	0.937729	...	1.046264	1.003663	0.353480
std	46.763983	68.166554	1.618632	...	0.403677	1.397240	0.806827
min	0.000000	0.000000	0.000000	...	0.281000	0.000000	0.000000
25%	15.790000	11.000000	0.000000	...	0.737000	0.000000	0.000000
50%	40.460000	42.683000	0.000000	...	1.020500	1.000000	0.000000
75%	70.022500	77.492750	1.000000	...	1.266500	2.000000	0.000000
max	347.320000	571.952000	18.000000	...	2.500000	11.000000	11.000000

Acute Aquatic Toxicity Prediction: Exploring Dataset

5

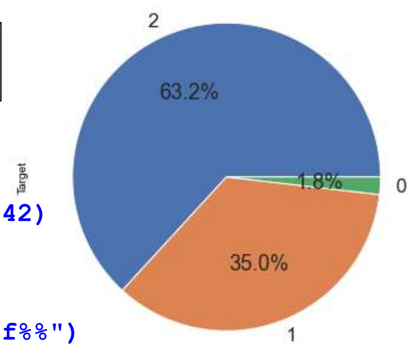
```
#Number of classes
c=dataset[class_col].nunique()
print("\nNumber of classes (discrete labels):", c)

#Number of instances per class
print("\nSample count per class:")
print(dataset[class_col].value_counts())
```

```
Number of classes (discrete labels): 3

Sample count per class:
2    345
1    191
0     10
Name: Target, dtype: int64
```

```
X = dataset[input_features]
y = dataset[class_col]
```

```
scaler = StandardScaler()
X=scaler.fit_transform(X)
```

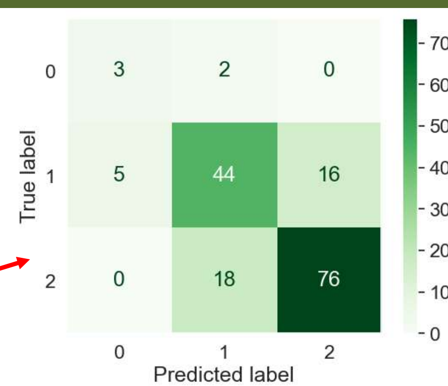
```
#split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size = 0.3, random_state = 42)

fig, ax=plt.subplots(1,1,figsize=(15,6))
sns.set(font_scale=1.5)
dataset['Target'].value_counts().plot.pie(autopct="%1.1f%%")
plt.show()
```



KNN Model

6

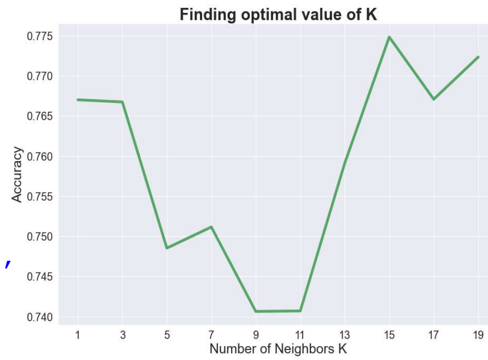
```
# Instantiate learning model (k = 3)
classifier = KNeighborsClassifier(n_neighbors=3)

# Fitting the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluating predictions
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
accuracy = accuracy_score(y_test, y_pred)*100
print(Accuracy of KNN model' + str(round(accuracy, 2)) + ' %.')
```




```
Accuracy of KNN model 75.0 %.
```

KNN Model

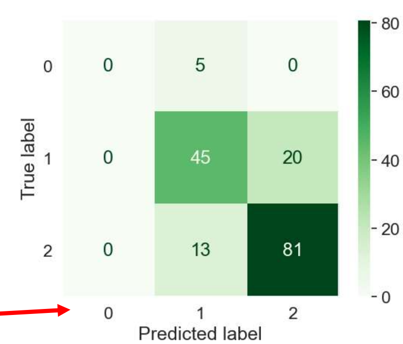
```
from sklearn.model_selection import cross_val_score
#creating list of K for KNN
k_list = list(range(1,20,2))
#creating list of cv scores
cv_scores = []

#performING 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train,
    cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

plt.figure(figsize=(15,10))
plt.title('Finding optimal value of K', fontsize=30, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=25)
plt.ylabel('Accuracy', fontsize=25)
plt.plot(k_list, cv_scores,linewidth=5,color='g')
plt.xticks(k_list,fontsize=20)
plt.yticks(fontsize=20)
```



KNN Model

```
best=k_list[cv_scores.index(max(cv_scores))]
print("Best value of K:",best)
# Instantiate learning model (k = best)
classifier = KNeighborsClassifier(n_neighbors=best)

# Fitting the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluating predictions
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```

```
Best value of K: 15
```



KNN Model

9

```
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of KNN model is ' + str(round(accuracy, 2)) + ' %.')

# A comprehensive classification report
report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", report)
```

```
Accuracy of KNN model is 76.83 %.
```

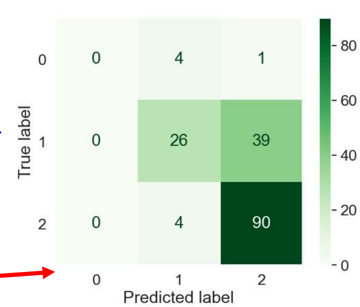
```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.71      0.69      0.70        65
           2       0.80      0.86      0.83        94

    accuracy                           0.77       164
   macro avg       0.51      0.52      0.51       164
weighted avg       0.74      0.77      0.75       164
```


Decision Tree Model

10

```
#check the shape of X_train and X_test
print("Traning set shape:",X_train.shape,"\nTest set shape:",X_test.shape)
# instantiate the DecisionTreeClassifier model with criterion entropy
clf_ent = DecisionTreeClassifier(criterion='entropy', max_depth=2,
random_state=0)

# fit the model
clf_ent.fit(X_train, y_train)
y_pred_ent = clf_ent.predict(X_test)

#Evaluating predictions
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred_ent)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels= np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```

```
Traning set shape: (382, 8)
Test set shape: (164, 8)
```



Decision Tree Model

11

```
# print the scores on training and test set
print('DT Training set score: {:.4f}'.format(clf_ent.score(X_train, y_train)))
print('DT Test set score: {:.4f}'.format(clf_ent.score(X_test, y_test)))
```

```
DT Training set score: 0.7487
DT Test set score: 0.7073
```

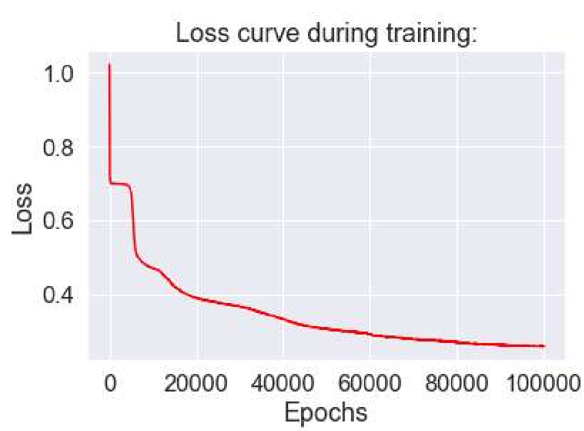
```
# A comprehensive classification report
report = classification_report(y_test, y_pred_ent)
print("\nDecision Tree Classification Report:\n", report)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.76      0.40      0.53        65
           2       0.69      0.96      0.80        94

    accuracy                           0.71       164
   macro avg       0.49      0.45      0.44       164
weighted avg       0.70      0.71      0.67       164
```


NN Model

12

```
# creating an classifier from the model:
mlp =MLPClassifier(hidden_layer_sizes = (5,4,3),activation = 'logistic',
solver = 'sgd',alpha = 0.01,learning_rate = 'adaptive',learning_rate_init=0.004,
max_iter = 100000,random_state = 42,tol = 0.000001,n_iter_no_change = 1000,
verbose=True)

# fit the training data to our model
mlp.fit(X_train, y_train)

# print("Loss curve during training:")
plt.figure()
plt.plot(range(1,len(mlp.loss_curve_)+1),
mlp.loss_curve_,color='red')
plt.grid(True)
plt.title("Loss curve during training:")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



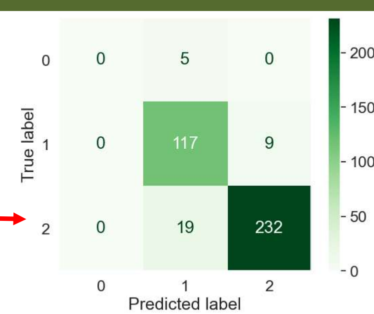
NN Model

```python
#Evaluating predictions
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

predictions_test = mlp.predict(X_test)
predictions_train = mlp.predict(X_train)

print('Training set score:')
print(accuracy_score(y_train,predictions_train))

print('Test set score:')
print(accuracy_score(y_test,predictions_test))

cm_train=confusion_matrix(y_train,predictions_train)
cm_pred=confusion_matrix(y_test,predictions_test)
```

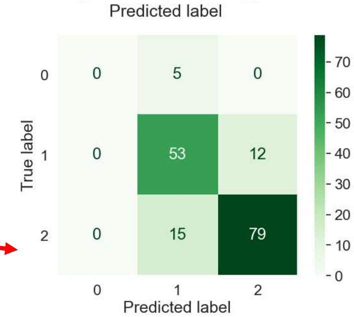
```
Training set score:
0.9136125654450262
Test set score:
0.8048780487804879
```


NN Model

```python
plt.figure()
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train,
display_labels= np.unique(y_train))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```




```python
plt.figure()
disp = ConfusionMatrixDisplay(confusion_matrix=cm_pred,
display_labels= np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```



NN Model

15

```
from sklearn.metrics import classification_report
print("Classification Report for Training:")
print(classification_report(y_train, predictions_train))

from sklearn.metrics import classification_report
print("Classification Report for Testing:")
print(classification_report(y_test, predictions_test))
```

```
Classification Report for Training:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.83      0.93      0.88       126
           2       0.96      0.92      0.94       251

    accuracy                           0.91       382
   macro avg       0.60      0.62      0.61       382
weighted avg       0.91      0.91      0.91       382
```

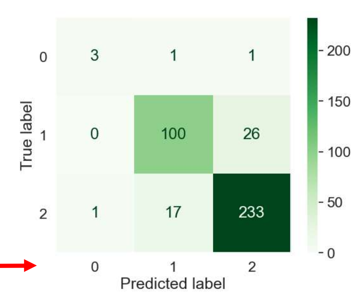
```
Classification Report for Testing:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.73      0.82      0.77        65
           2       0.87      0.84      0.85        94

    accuracy                           0.80       164
   macro avg       0.53      0.55      0.54       164
weighted avg       0.79      0.80      0.79       164
```


SVM Model

16

```
classifier = SVC(C=10,kernel='rbf',gamma='scale')
classifier.fit(X_train, y_train)

# Now predict the value of the test
predictions_test = classifier.predict(X_test)
predictions_train = classifier.predict(X_train)
print('Training set score:')
print(accuracy_score(y_train,predictions_train))
print('Test set score:')
print(accuracy_score(y_test,predictions_test))

cm_train=confusion_matrix(y_train,predictions_train)
cm_pred=confusion_matrix(y_test,predictions_test)
plt.figure()
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train,
display_labels= np.unique(y_train))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```

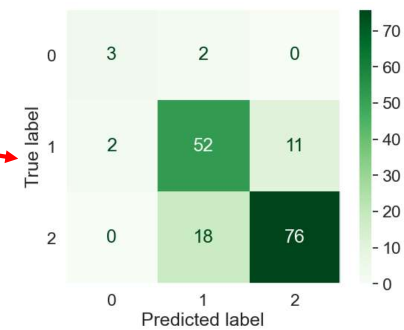
```
Training set score:
0.8795811518324608
Test set score:
0.7987804878048781
```



SVM Model

17

```
plt.figure()
disp = ConfusionMatrixDisplay(confusion_matrix=cm_pred,
display_labels= np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```




```
from sklearn.metrics import classification_report
print("Classification Report for Training:")
print(classification_report(y_train, predictions_train))

from sklearn.metrics import classification_report
print("Classification Report for Testing:")
print(classification_report(y_test, predictions_test))
```


SVM Model

18

on Training Samples

```
Classification Report for Training:
              precision    recall  f1-score   support

           0       0.75      0.60      0.67         5
           1       0.85      0.79      0.82       126
           2       0.90      0.93      0.91       251

    accuracy                           0.88       382
   macro avg       0.83      0.77      0.80       382
weighted avg       0.88      0.88      0.88       382
```

on Test Samples

```
Classification Report for Testing:
              precision    recall  f1-score   support

           0       0.60      0.60      0.60         5
           1       0.72      0.80      0.76        65
           2       0.87      0.81      0.84        94

    accuracy                           0.80       164
   macro avg       0.73      0.74      0.73       164
weighted avg       0.81      0.80      0.80       164
```

Naïve Bayes Model

19

```
#created a object for the classifier
NB=GaussianNB()
#train using train data
NB.fit(X_train,y_train)

from sklearn.metrics import
confusion_matrix,ConfusionMatrixDisplay,classification_report,accuracy_score
import numpy as np

#predict target data for test feature data and save into pred variable
predicted=NB.predict(X_test)
print("Test Accuracy={:.4f}".format(accuracy_score(y_test, predicted)))

predicted_train=NB.predict(X_train)
print("Train Accuracy={:.4f}".format(accuracy_score(y_train,
predicted_train)))
```

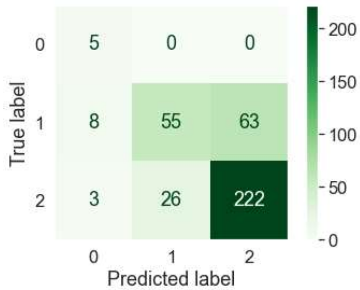
```
Test Accuracy=0.7134
Train Accuracy=0.7382
```

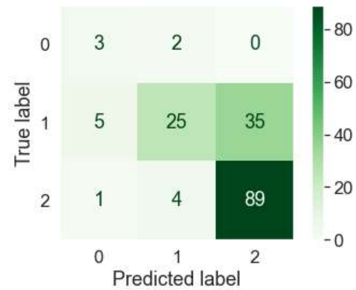

Naïve Bayes Model

20

```
print("Classification report Training:\n",classification_report(y_train,
predicted_train))
cm_pred = confusion_matrix(y_train, predicted_train)
plt.figure()
disp =
ConfusionMatrixDisplay(confusion_matrix=cm_pred,display_labels=np.unique(y_train))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()

print("Classification report Test:\n",classification_report(y_test, predicted))
cm_pred = confusion_matrix(y_test, predicted)
plt.figure()
disp = ConfusionMatrixDisplay(confusion_matrix=cm_pred,
display_labels=np.unique(y_test))
disp.plot(cmap='Greens')
plt.grid(False)
plt.show()
```

Naïve Bayes Model






```
Classification report Training:
              precision    recall  f1-score   support

           0       0.31      1.00      0.48         5
           1       0.68      0.44      0.53       126
           2       0.78      0.88      0.83       251

    accuracy                           0.74       382
   macro avg       0.59      0.77      0.61       382
weighted avg       0.74      0.74      0.73       382
```

```
Classification report Test:
              precision    recall  f1-score   support

           0       0.33      0.60      0.43         5
           1       0.81      0.38      0.52        65
           2       0.72      0.95      0.82        94

    accuracy                           0.71       164
   macro avg       0.62      0.64      0.59       164
weighted avg       0.74      0.71      0.69       164
```


Questions?