Bangladesh University of Business and Technology

Department of CSE

Assignment -03

Course Title : Data Mining
Course code : CSE-476

Submitted By

Ishmoth Ura Nuri
ID : 18192103237
Intake: 41
Section: 03

Submitted To

Khan Md. Hasib
Lecturer
Computer Science & Engineering

Submission Date : 22.02.23

What are the differences between the following data sets?
1- Contact Lens data: http://archive.ics.uci.edu/ml/datasets/Lenses
2- Iris data: http://archive.ics.uci.edu/ml/datasets/Iris
From the following algorithms which one is expected to perform best on the Contact Lens data? Implement those on the Contact Lens data→ K-Nearest Neighbors
Decision Tree
Neural Networks

Ans : The Lenses and Iris datasets are both commonly used in machine learning and statistical analysis, but they differ in several important ways:

Data Type: The Lenses dataset consists of categorical data, while the Iris dataset contains continuous data. Specifically, the Lenses dataset has four input attributes, each with three possible values, and a binary class attribute. The Iris dataset, on the other hand, has four continuous input attributes and a multi-class class attribute.

Size and Dimensions: The Iris dataset is larger than the Lenses dataset, with 150 instances versus 24 instances. The Iris dataset also has more dimensions, with four input attributes versus three input attributes in the Lenses dataset.
Application: The two datasets are typically used for different applications. The Lenses dataset is often used for classification tasks, such as predicting whether a patient needs to wear glasses based on their eye exam results. The Iris dataset is commonly used for clustering tasks, such as identifying distinct groups of iris flowers based on their characteristics.
Difficulty: The Lenses dataset is considered easier to analyze than the Iris dataset. This is due in part to the categorical nature of the Lenses dataset, which makes it simpler to interpret and analyze than the continuous data in the Iris dataset.

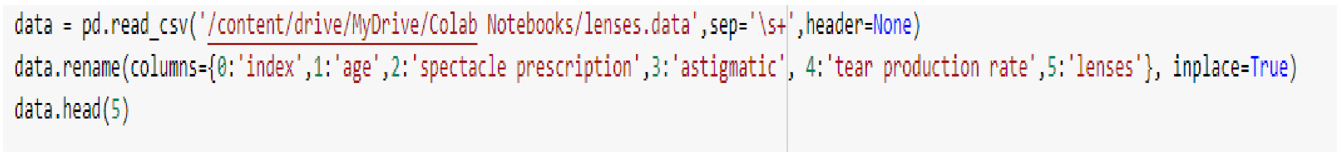
It is difficult to determine which algorithm is expected to perform best on the Contact Lens data without more information about the specific characteristics of the dataset and the objectives of the analysis.

Decision Trees: Decision trees are often used for classification tasks and are particularly useful for datasets with categorical input variables. They can handle non-linear relationships and are relatively easy to interpret, making them a popular choice for datasets like Contact Lens.

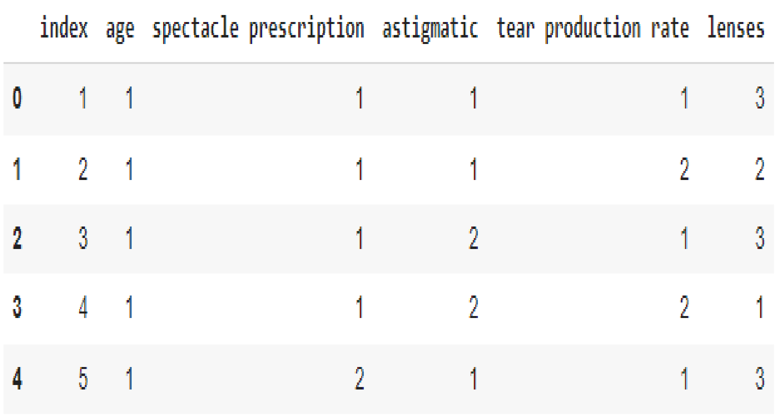
Neural networks : It can also be effective on this dataset as they can handle non-linear relationships between the input features and output labels. However, for small datasets like this, overfitting can be a problem if the model is too complex or the number of training examples is too small.
k-Nearest Neighbors (KNN): KNN is a non-parametric algorithm that can be used for both classification and regression tasks. It works by finding the k closest instances in the training set to a new instance and using their class labels to predict the label of the new instance. It can work well for small datasets like Contact Lens.

So ,the best approach would be to try all three algorithms and evaluate their performance using appropriate metrics such as accuracy, precision, recall, and F1-score. Cross-validation can also be used to estimate the performance of each model and reduce the risk of overfitting.

1. Upload the dataset and assign column names and showing the top 5 values

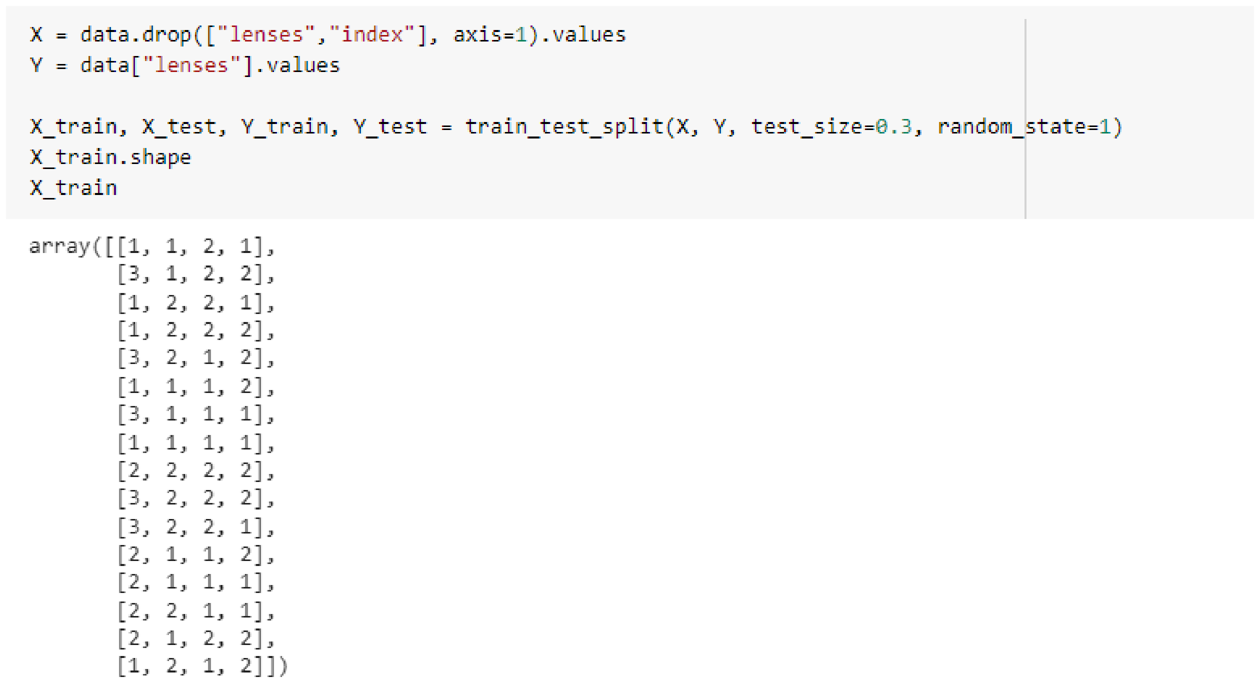


```
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/lenses.data',sep='\s+',header=None)
data.rename(columns={0:'index',1:'age',2:'spectacle prescription',3:'astigmatic', 4:'tear production rate',5:'lenses'}, inplace=True)
data.head(5)
```



	index	age	spectacle prescription	astigmatic	tear production rate	lenses
0	1	1	1	1	1	3
1	2	1	1	1	2	2
2	3	1	1	2	1	3
3	4	1	1	2	2	1
4	5	1	2	1	1	3

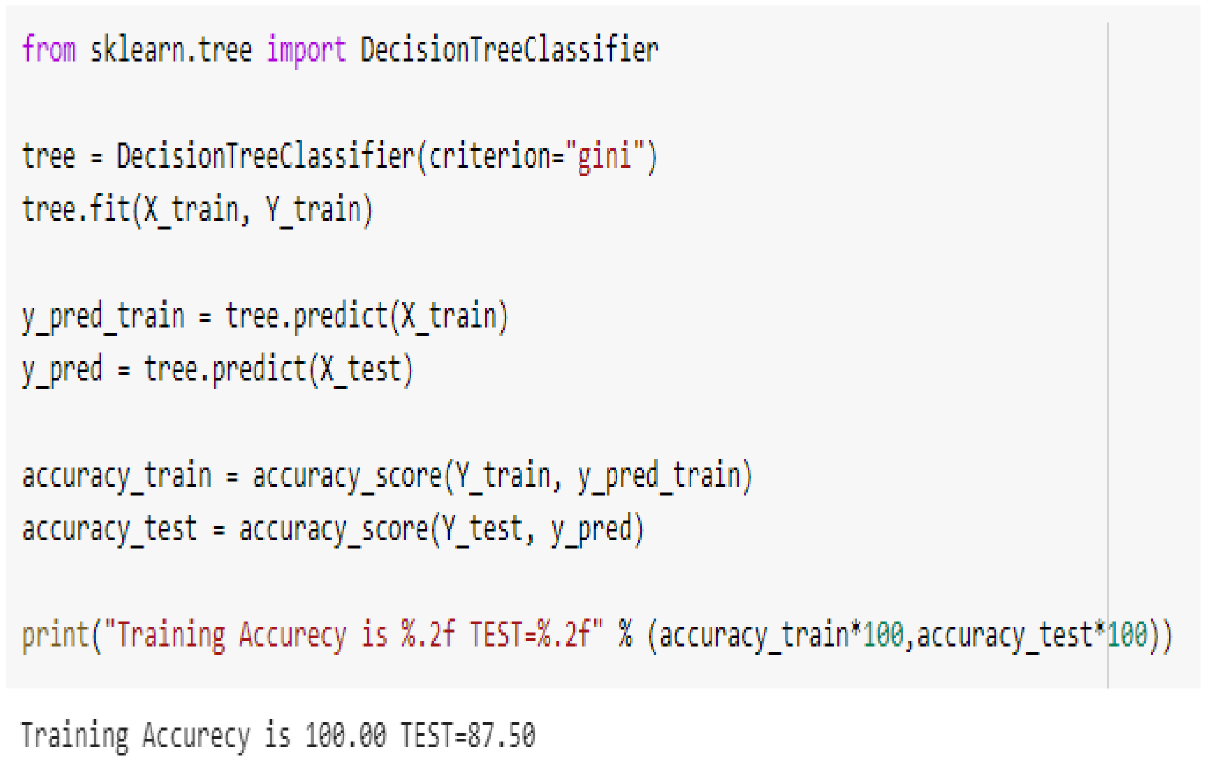
2. Splitting the data into train & test data



```
X = data.drop(["lenses","index"], axis=1).values
Y = data["lenses"].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
X_train.shape
X_train
```

```
array([[1, 1, 2, 1],
       [3, 1, 2, 2],
       [1, 2, 2, 1],
       [1, 2, 2, 2],
       [3, 2, 1, 2],
       [1, 1, 1, 2],
       [3, 1, 1, 1],
       [1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 2, 2, 2],
       [3, 2, 2, 1],
       [2, 1, 1, 2],
       [2, 1, 1, 1],
       [2, 2, 1, 1],
       [2, 1, 2, 2],
       [1, 2, 1, 2]])
```

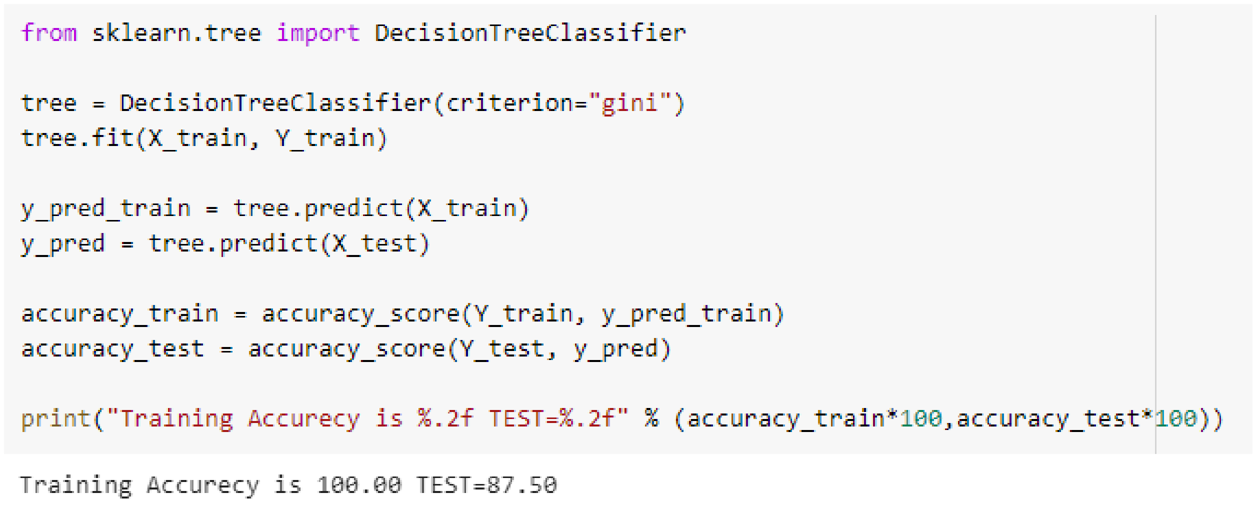
3. Applying the Decision Tree to the lens dataset



```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion="gini")
tree.fit(X_train, Y_train)

y_pred_train = tree.predict(X_train)
y_pred = tree.predict(X_test)

accuracy_train = accuracy_score(Y_train, y_pred_train)
accuracy_test = accuracy_score(Y_test, y_pred)

print("Training Accurecy is %.2f TEST=%.2f" % (accuracy_train*100,accuracy_test*100))
```

```
Training Accurecy is 100.00 TEST=87.50
```

4. After the training, we got the training Accuracy = 100% & testing accuracy= 87%

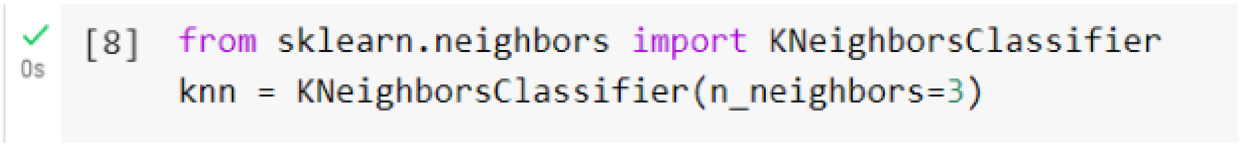


```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion="gini")
tree.fit(X_train, Y_train)

y_pred_train = tree.predict(X_train)
y_pred = tree.predict(X_test)

accuracy_train = accuracy_score(Y_train, y_pred_train)
accuracy_test = accuracy_score(Y_test, y_pred)

print("Training Accurecy is %.2f TEST=%.2f" % (accuracy_train*100,accuracy_test*100))
```

```
Training Accurecy is 100.00 TEST=87.50
```

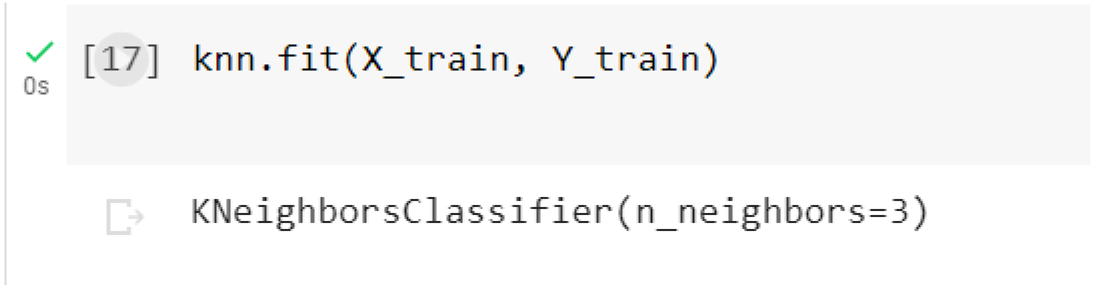
Applying K-Nearest Neighbors

1. Importing the KNeighborsClassifier library & also defining the k-neighbors model.



```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
```

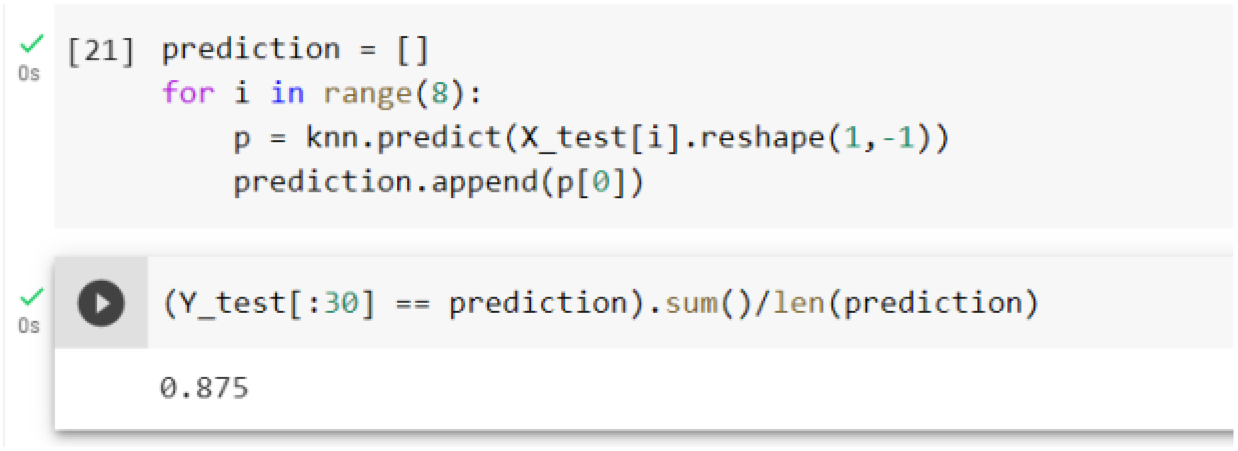
2. Fitting the dataset in the KNN Model.



```
knn.fit(X_train, Y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

3. Storing the predicted values and here we got an accuracy of 87%



```
[21] prediction = []
     for i in range(8):
         p = knn.predict(X_test[i].reshape(1,-1))
         prediction.append(p[0])
```

```
(Y_test[:30] == prediction).sum()/len(prediction)
```

```
0.875
```