

Experiment-2

Implement using an open-source dataset

Aim: To implement a supervised machine learning model using an open-source dataset.

Pseudocode:

- 1) import necessary libraries
 - pandas, scikit-learn, dataset, metrics, KNeighboursClassifier.
- 2) Load the dataset
 - use datasets.load-iris()
- 3) Prepare the data
 - Assign features to x target to y.
- 4) Split into training and testing sets
 - use train_test_split (X, y, test-size = 0.2, random-state = 42)
- 5) Instantiate the KNN classifier:
 - KNN = KNeighboursClassifier (n-neighbours = 3)
- 6) Train the model
- 7) Make predictions
- 8) Evaluation the classifier:
 - Calculate accuracy: metrics.accuracy_score(y.test, y-pred)

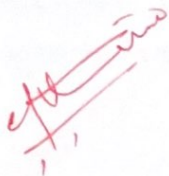
Observations:

- KNN classifier is trained on the Iris dataset and tested with unseen data
- Output is displayed
- lowering 'k' can make the model even more sensitive to noise, while larger k can smoothen decision boundaries

Result:

Accuracy: 1.00

Predicted class for sample $[5.1, 3.5, 1.4, 0.2]$: setosa



File	Edit	View	Run	Kernel	Tools	Settings	Help
Filter files by name							
/ Deep Learning /							
Name				Last Modified			
Lab_2.ipynb				next year			
Lab_3.ipynb				next year			

```
[5]: import numpy as np
import pandas as pd
import sklearn
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
[6]: df = load_iris()
```

```
[7]: X=df.data
y=df.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
```

```
[9]: knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
sample = [[5.1, 3.5, 1.4, 0.2]]
predicted_class = df.target_names[knn.predict(sample)[0]]
print(f"Predicted class for sample {sample}: {predicted_class}")
```

```
Accuracy: 1.00
Predicted class for sample [[5.1, 3.5, 1.4, 0.2]]: setosa
```


Study of classifiers with respect to statistical parameters

Aim: To study the performance of different classifiers using statistical parameters like accuracy, precision, recall, & f1 score.

Description:

1. Accuracy -

The ratio of correctly predicted instances to the total instances in the dataset.

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{total predictions}}$$

2. Precision -

The ratio of correctly predicted positive instances to all instances predicted as positive

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False Positives}}$$

3. Recall -

The ratio of correctly predicted positive instances to all actual positive instances. It measures how well the model identifies actual positives.

$$\text{recall} = \frac{\text{True Positives}}{\text{True positives} + \text{False negatives}}$$

4. F1 Score -

The harmonic mean of precision and recall, providing a balance between the two metrics. It is a single metric used to evaluate the trade-off between precision & recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5 Confusion Matrix -

This is a table showing the correct and incorrect predictions across classes. It helps visualize model performance with True Positive (TP), False Positive (FP), False Negatives (FN), and True Negatives (TN).

Procedure :

- (i) Load the open source dataset
- (ii) Split dataset into training and testing sets
- (iii) Train the classifier
- (iv) Predict labels on test data
- (v) Evaluate each classifier, using accuracy, precision, etc.
- (vi) Visualize the confusion matrix

Observation:

Classifier	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.93	0.98	0.98	0.98
KNN	0.96	0.98	0.99	0.97
Decision Tree	0.94	0.97	0.94	0.95

Logistic Regression:

61(TP)	2(FN)
2(FP)	106(TN)

where malignment
(conceals)

↓
+ve

Benign (non-concave)

↓
-ve

KNN

57(TP)	6(FN)
1(FP)	107(TN)

Decision Tree

60(TP)	3(FN)
7(FP)	101(TN)

Result:

The classification of logistic regression, KNN and Decision tree were successfully implemented. All models achieved high scores with logistic regression having the highest overall performance.

~~11/14/18/20~~

Filter files by name		
Name	Last Modified	
data	3 months ago	
januarthc	6 months ago	
nnml	4 months ago	
on	3 months ago	

```

)
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"Model: {name}")
    print(f"Accuracy: ", accuracy_score(y_test, y_pred))
    print(f"Precision: ", precision_score(y_test, y_pred))
    print(f"Recall: ", recall_score(y_test, y_pred))
    print(f"F1 Score: ", f1_score(y_test, y_pred))
    print(f"Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("-" * 30)

```

```

Model: LogisticRegression
Accuracy: 0.97668371345029
Precision: 0.984814814814815
Recall: 0.9834814814814815
F1 Score: 0.98414814814815
Confusion Matrix:
[[ 61  2]
 [ 2 106]]

```

```

Model: svm
Accuracy: 0.959643374653881
Precision: 0.946033546572657
Recall: 0.9907467467467467
F1 Score: 0.9683257938532935
Confusion Matrix:
[[ 57  6]
 [ 3 107]]

```

```

Model: decisionTree
Accuracy: 0.929824544635886
Precision: 0.952881868792453
Recall: 0.9310518181818181
F1 Score: 0.9439252364445598
Confusion Matrix:
[[ 58  9]
 [ 7 101]]

```