MSPA-2

(EXPERIMENT 4 to EXPERIMENT 6)

Name – vinay partap singh

Course Name – MLT Lab

Section – B

Course Code – CSE-2424

Roll No. – 177

Reg No. – 20010232

EXPERIMENT NO. 4

Name of experiment –

Implementing a Classification model using KNN algorithm from Scikit learn library.

Problem Statement –

A hobby botanist is interested in distinguishing the species of some iris flowers that she has found.She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimetres. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica.For these measurements, she can be certain of which species each iris belongs to. Let's assume that these are the only species our hobby botanist will encounter in the wild.Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.

Theory –

K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm used for classification tasks. It operates on the principle that similar data points tend to belong to the same class. Here's a concise explanation in 200 words:

KNN classifies data points based on their proximity to other data points in a feature space. The "K" in KNN refers to the number of nearest neighbors to consider when making a prediction. To classify a new data point, KNN calculates the distances between the point and its K-nearest neighbors in the training dataset, typically using Euclidean distance.

Once these distances are computed, the algorithm assigns the majority class among the K-nearest neighbors to the new data point. In other words, it looks at the class labels of the K-nearest neighbors and chooses the most common class as the predicted class for the new point. This makes KNN a "lazy learner" since it doesn't build a model during training but rather memorizes the training data.

KNN's simplicity is both its strength and weakness. It's easy to implement and can handle complex decision boundaries. However, its performance can be sensitive to the choice of K and the distance metric used. Additionally, it can be computationally expensive for large datasets since it requires calculating distances for each data point.

In summary, KNN is a versatile algorithm for classification, particularly useful when the decision boundary is not well-defined. It's essential to fine-tune K and choose an appropriate distance metric to achieve optimal results in practice.

Data Description–

Keys of iris_dataset:

'feature_names' 'target' 'frame' 'DESCR' 'target_names 'filename' 'data' .

Data Set Characteristics -

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information –

• sepal length in cm

• sepal width in cm

• petal length in cm

• petal width in cm

• class:

  o Iris-Setosa

  o Iris-Versicolour

  o Iris-Virginica

Feature Names-

'sepal length (cm)' 'sepal width (cm)'

'petal length (cm)' 'petal width (cm)

Target Names-

'setosa'	'versicolor'	'virginica'

Shape of data: (150, 4)

Problem Analysis-

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between Lengths and Widths of Petals and Sepals and the category of iris.

This type of problem can easily be dealt with by using K Nearest Neighbours for Classification.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small).

If k = 1, then the object is simply assigned to the class of that single nearest neighbour.

Machine Learning Task- Classification

Predictors-

'sepal length (cm)' 'sepal width (cm)'

'petal length (cm)' 'petal width (cm)

Response Variable- EITHER OF THE THREE FLOWER SPECIES:

'setosa'	'versicolor'	'virginica'

- Model Parameters: Mean, median
- Hyper Parameters:
  - Value of K
  - Distance Metric (like 'euclidean','minkowski','manhattan',etc.)

Pseudocode-

- Load the data
- Initialize the value of k
- Calculate the distance between test data and each row of training data.
- Sort the calculated distances in ascending order based on distance values
- Get top k rows from the sorted array
- Get the most frequent class of these rows as predicted class

Program –

```
import numpy as np
import pandas as pd
#Loading the dataset
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

```
data = load_iris()
X_train, X_test, Y_train, Y_test = train_test_split(
data.data, data.target,
stratify=data.target,test_size=0.25)
# Hyperparameter
param={
    'n_neighbors':list(range(1,12,2)),
    'metric':['euclidean','minkowski','manhattan'],
    'weights':['uniform','distance']
}
print("Grid Result For Five Best Parameters: ")
clf=GridSearchCV(KNeighborsClassifier(),param_grid=param,cv=5,return_train
_score=False)
clf.fit(X_train,Y_train)
result=pd.DataFrame(clf.cv_results_)[['param_n_neighbors','param_metric','
param_weights','mean_test_score']]
print(result.sort_values(by='mean_test_score',ascending=False).head(5))
print(f'Best Parameters for KNN: {clf.best_params_} \n with score:
{clf.best_score_}')
#Create KNN Classifiers
a=clf.best_params_
knn = KNeighborsClassifier(n_neighbors=a['n_neighbors'],
                           weights=a['weights'],metric=a['metric']);
#Train the classifier model using the training set
knn.fit(X_train, Y_train)
#Predict the response for test data x_new(dimensions of the new iris
flower to be classified)
x_new = np.array([[5, 2.9, 1, .2]])
y_pred = knn.predict(x_new)
print(f"\nTest Data: {x_new} Prediction: {y_pred}")
print(f"Predicted target name: {data['target_names'][y_pred]}")
# MODEL EVALUATION: Predict the responses for test dataset and calculate
model accuracy
# Model Accuracy: how often is the classifier correct?
Y_pred = knn.predict(X_test)
print(f"\nTest set predictions:\n {Y_pred}")
```

Output –

```
Grid Result For Five Best Parameters:
   param_n_neighbors param_metric param_weights  mean_test_score
2                  3    euclidean       uniform         0.973518
3                  3    euclidean      distance         0.973518
15                 3    minkowski      distance         0.973518
14                 3    minkowski       uniform         0.973518
0                  1    euclidean       uniform         0.964427
Best Parameters for KNN: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}
 with score: 0.9735177865612649

Test Data: [[5.  2.9 1.  0.2]] Prediction: [0]
Predicted target name: ['setosa']

Test set predictions:
 [0 0 2 0 2 2 1 1 0 2 0 1 0 0 0 1 2 0 1 2 2 2 0 2 0 1 1 2 1 1 2 0 1 2 0 1 1
 2]
```

Conclusion – In this experiment we learnt about how to create a KNN (K-Nearest Neighbour) model using sickit-learn and implement on the iris flower dataset.

EXPERIMENT NO. 5

Name of experiment –

KNN for regression implemented using KNN algorithm from Scikit learn

library.

Problem Statement –

Given a set of features that describe a house in Boston, design an optimal KNN regression model that can predict the house price for any given house.

Theory –

K-Nearest Neighbors (KNN) is a versatile machine learning algorithm primarily used for regression tasks. Unlike its classification counterpart, KNN for regression predicts a continuous output variable based on the similarity of input data points.

In KNN regression, the "K" refers to the number of nearest neighbors that influence the prediction. To make a prediction for a new data point, the algorithm identifies the K nearest data points in the training dataset based on a chosen distance metric, typically Euclidean distance. These neighbors' output values are then averaged (for a simple regression) or weighted (for weighted regression) to estimate the target value for the new point.

KNN regression is intuitive and can capture complex relationships between input and output variables. However, it's essential to choose the appropriate value of K, as a small K may lead to overfitting, while a large K might result in underfitting. The distance metric selection is also crucial, as it affects the algorithm's sensitivity to different feature scales.

In summary, KNN regression is a straightforward yet effective technique for regression tasks, relying on the proximity of data points to make predictions. Careful tuning of the K value and distance metric is necessary to optimize its performance for specific datasets and applications.

Data Description– The Boston Housing Dataset is derived from information collected by the U.S. Census Service concerning housing in the area of Boston, MA. There are 506 observations with 13 features (independent variables) like: the number of rooms (rm), crime rate (crim), air pollution variable (nox), cost of public services in each community (tax), pupil-teacher ratio (ptratio), etc. The dependent/target variable is house price, which is given in thousand dollars.

Problem Analysis-

Upon critical analysis of the given problem, it can be seen that there's a relationship between the price of the house and the different features given in the dataset description.This type of problem can easily be dealt with by using K Nearest Neighbours for Regression.

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors. KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood.

Machine Learning Task- Regression.

Predictor Variables-

- CRIM: Per capita crime rate by town.
- ZN: Proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS: Proportion of non-retail business acres per town.
- CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise).
- NOX: Nitric oxides concentration (parts per 10 million).
- RM: Average number of rooms per dwelling.
- AGE: Proportion of owner-occupied units built prior to 1940.

- DIS: Weighted distances to five Boston employment centers.
- RAD: Index of accessibility to radial highways.
- TAX: Full-value property-tax rate per $10,000.
- PTRATIO: Pupil-teacher ratio by town.
- B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
- LSTAT: % lower status of the population.

Response Variable- The price of the house in Thousand Dollars.

Model Parameters- Final cluster centers.

Hyper Parameters-

- Value of K.
- Distance Metric (e.g., 'euclidean', 'minkowski', 'manhattan', etc.)
- Seed/initial values for cluster center.

Pseudocode for Elbow Method-

```
fun elbow(): return elbow point of the graph
dataset = load_boston()
split data into training and testing
for k from 1 to sqrt(no. of rows in dataset):
   model = KNeighborsRegressor(k)
   model.fit(data_train)

   pred = model.predict(data_test)
```

Pseudo Code for KNN -

- Load the data
- Initialize the value of k

- Calculate the distance between test data and each row of training data.
- Sort the calculated distances in ascending order based on distance values.
- Get the top k rows from the sorted array.
- Get the mean of these rows as the predicted value.

Program –

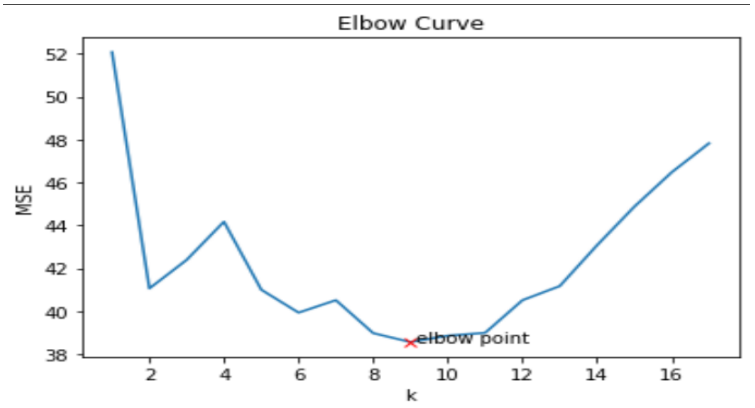
```
from sklearn.datasets import load_boston
from sklearn import neighbors
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import math
import warnings

warnings.filterwarnings(action='ignore')

# Loading the Boston dataset
boston = load_boston()
x = boston.data[:, :]
y = boston.target
print(x.shape, y.shape)

tsize = 0.30  # 30% of total data is used for testing and 70% used for
training

# Splitting the dataset into training and testing sets
# (parameter random state is fixed at some integer, to ensure the same
train and test sets across various runs)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=tsize,
random_state=102)

# Code using Scikit-learn library for KNN regression
# Finding MSEs for different values of k
maxk = int(math.sqrt(xtrain.shape[0]))  # Maximum value of k
mse_val = []  # To store RMSE values for different k

for K in range(1, maxk):
    model = neighbors.KNeighborsRegressor(n_neighbors=K)
    model.fit(xtrain, ytrain)  # Fit the model
    pred = model.predict(xtest)  # Make prediction on the test set
    error = mean_squared_error(ytest, pred)  # Calculate RMSE
```

```
    mse_val.append(error)  # Store RMSE values
    print('MSE value for k = ', K, 'is:', error)

# Function to find the elbow point
def find_elbow():
    inds = np.argsort(mse_val)
    for i in inds:
        diff1 = mse_val[i - 1] - mse_val[i]
        diff2 = mse_val[i] - mse_val[i + 1]
        if (diff1 > 0 and diff2 < 0):
            break
    eb1 = i + 1
    return eb1

# Plotting the elbow curve
k = np.arange(1, maxk)
xl = "k"
yl = "MSE"
plt.xlabel(xl)
plt.ylabel(yl)
plt.title("Elbow Curve")
plt.plot(k, mse_val)

# Finding the k for the elbow point
ke = find_elbow()
print("Best Value of k using elbow curve is", ke)
plt.plot(ke, mse_val[ke - 1], 'rx')
plt.annotate(" elbow point", (ke, mse_val[ke - 1]))

# Now with the best k(i.e. ke), predict the cost for the new house with
given features
xnew = np.array([2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00,
4.6900e-01, 6.4210e+00, 7.8900e+01,
                 4.9671e+00, 2.0000e+00, 2.4200e+02, 1.7800e+01,
3.9690e+02, 9.1400e+00])
model = neighbors.KNeighborsRegressor(n_neighbors=ke)
model.fit(xtrain, ytrain)  # Fit the model
xnew = xnew.reshape(1, -1)
hcost = model.predict(xnew)
print("Predicted price of the given house is {:.2f}".format(hcost[0]),
"thousand dollars")

plt.show()
```

Output –

```
(506, 13) (506,)
MSE value for k =  1 is: 52.069736842105264
MSE value for k =  2 is: 41.056726973684206
MSE value for k =  3 is: 42.38861842105264
MSE value for k =  4 is: 44.16745065789473
MSE value for k =  5 is: 40.99719736842106
MSE value for k =  6 is: 39.929205043859646
MSE value for k =  7 is: 40.511697099892594
MSE value for k =  8 is: 38.973344983552636
MSE value for k =  9 is: 38.55725064977258
MSE value for k =  10 is: 38.853878289473684
MSE value for k =  11 is: 38.987844715093516
MSE value for k =  12 is: 40.50879797149122
MSE value for k =  13 is: 41.166428682653375
MSE value for k =  14 is: 43.0698419038668
MSE value for k =  15 is: 44.86796023391812
MSE value for k =  16 is: 46.46832879317434
MSE value for k =  17 is: 47.82060599162265
Best Value of k using elbow curve is 9
Predicted price of the given house is 24.00 thousand dollars
```

Graph –



Conclusion – In this experiment we learnt about KNN regression using the scikit-learn and implemented on boston housing data.

EXPERIMENT NO. 6

Name of experiment –

Naïve Bayes Classifier using Scikit learn library on Wine Dataset.

Problem Statement –

Wine classification is a very famous multi-class classification problem. The Wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The Dataset comprises of 13 features (alcohol, malic_acid, ash, alcalinity_of_ash, magnesium, total_phenols, flavanoids, nonflavanoid_phenols, proanthocyanins, color_intensity, hue, od280/od315_of_diluted_wines, proline) and type of wine cultivar. This data has three types of wine Class_0, Class_1, and Class_3. Build a ML model to classify the type of wine.

Theory –

Naive Bayes is a popular machine learning algorithm used for classification and probabilistic modeling. It's based on Bayes' theorem, which is a fundamental concept in probability theory and statistics. The "naive" aspect comes from the simplifying assumption that the features used to describe data are conditionally independent, given the class label. This assumption simplifies the calculation of probabilities and makes the algorithm computationally efficient.

Mathematically, Naive Bayes can be expressed as:

$P(C|X) = (P(C) * P(X|C))/P(X)$

Where:

- $(P(C|X))$ is the posterior probability of class (C) given features X.

- $(P(C))$ is the prior probability of class (C).

- $(P(X|C))$ is the likelihood of observing features (X) given class C.

- $(P(X))$ is the probability of observing features (X).

In practice, you calculate the posterior probability for each possible class and choose the class with the highest probability as the predicted class.

For text classification, like spam detection, the features are often word occurrences or frequencies. The "naive" assumption means that we assume the words are independent within a document, even though this may not be entirely true.

Despite its simplicity, Naive Bayes can perform surprisingly well, especially for text classification tasks. It's computationally efficient, making it suitable for large datasets, and it serves as a baseline model for more complex algorithms like decision trees or neural networks.

Data Description–

```
#   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  classes                       178 non-null    int32
```

Program –

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix

# Load the Wine dataset
wine = datasets.load_wine()

# Print the names of the 13 features
print("\nFeatures: ",
wine.feature_names[0:7],'\n',wine.feature_names[7:13])

# Print the label types of wine (class_0, class_1, class_2)
print("\nLabels: ", wine.target_names)

# Print the dimensions of wine data
print("\nShape of data:", wine.data.shape)
```

```
# Print the wine data features (first 5 records)
print("\nFirst five records:\n", wine.data[0:5])

# Print the first five targets
print("\nFirst five Targets:", wine.target[0:5])

# Split the data into training and testing (70% training and 30% test) and
show class-wise distribution
X_train, X_test, y_train, y_test = train_test_split(wine.data,
wine.target, test_size=0.3, stratify=wine.target)

# Print class-wise distribution of observations in the whole set
print('\nClass distribution for wine dataset: %s' %
np.bincount(wine.target))

# Print class-wise distribution of observations in the training set
print('Class distribution for training data: %s' % np.bincount(y_train))

# Print class-wise distribution of observations in the test set
print('Class distribution for test data: %s' % np.bincount(y_test))

# Create a Gaussian Naive Bayes classifier
gnb = GaussianNB()

# Train the model using the training sets
gnb.fit(X_train, y_train)

# Predict the response for the test dataset
y_pred = gnb.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("\nAccuracy:", metrics.accuracy_score(y_test, y_pred))

# Display the confusion matrix
print('Confusion Matrix is:\n', confusion_matrix(y_test, y_pred))

# Display the classification report
print('\nClassification Report:\n', metrics.classification_report(y_test,
y_pred))

# Predict the response for a new test data point
x_new = np.array([wine.data[0]])
y_pred = gnb.predict(x_new)

print("\nNew Test Data: {}".format(x_new))
print("Prediction: Class {}".format(y_pred))
print("Predicted target name: {}".format(wine['target_names'][y_pred]))
```

Output –

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids']
 ['nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

Labels:  ['class_0' 'class_1' 'class_2']

Shape of data: (178, 13)

First five records:
 [[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
  3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
  2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
 [1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
  3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]]

First five Targets: [0 0 0 0 0]

Class distribution for wine dataset: [59 71 48]
Class distribution for training data: [41 50 33]
Class distribution for test data: [18 21 15]
```

```
Accuracy: 0.9629629629629629
Confusion Matrix is:
 [[18  0  0]
 [ 0 19  2]
 [ 0  0 15]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.90      0.95        21
           2       0.88      1.00      0.94        15

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54


New Test Data: [[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]]
Prediction: Class [0]
Predicted target name: ['class_0']
```

Conclusion – In this experiment we learnt about Naïve Bayes and how to implement it on wine data to classify wine class.