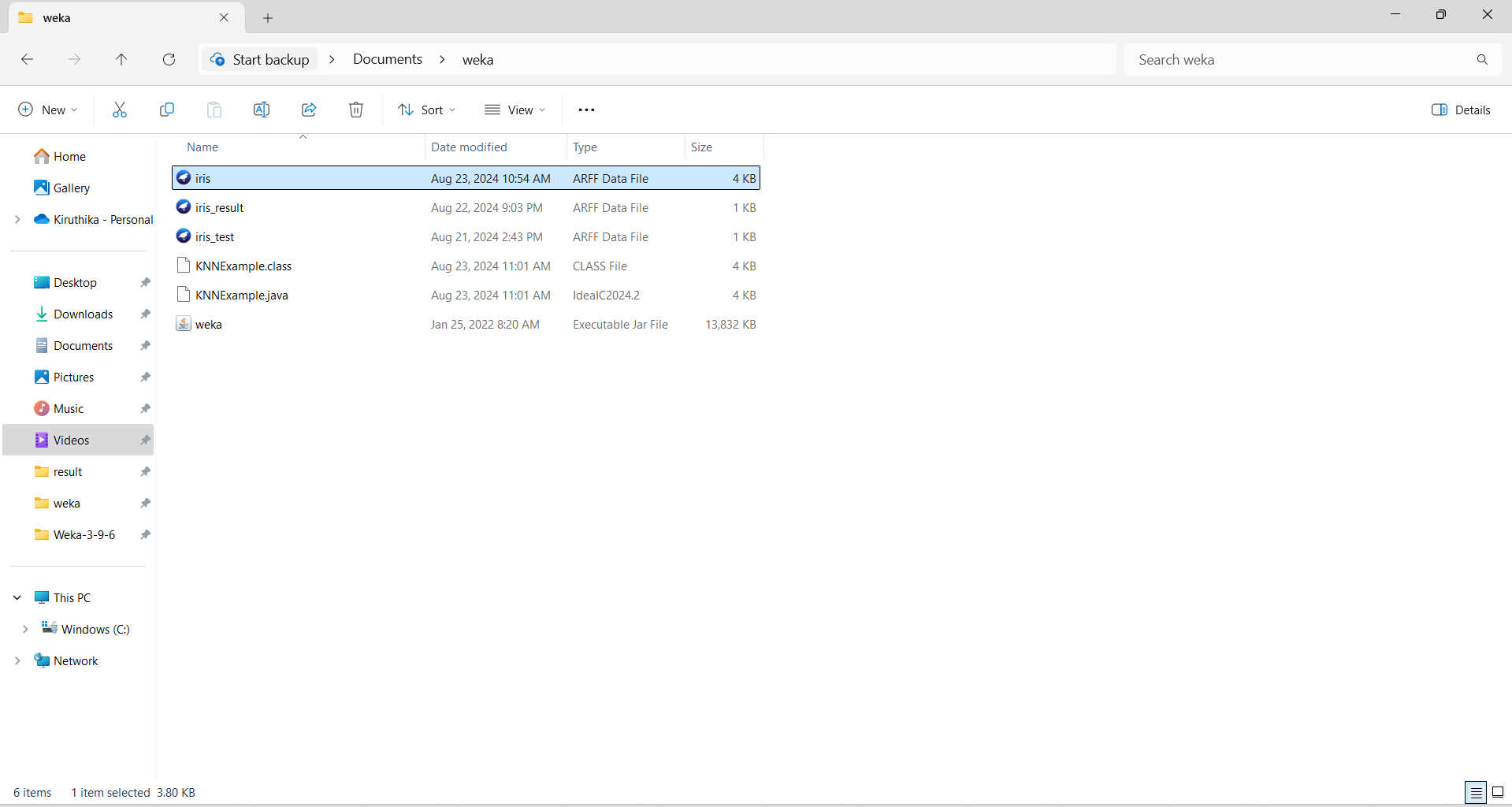
**Weka implementation in java.**

**KNN Algorithm**

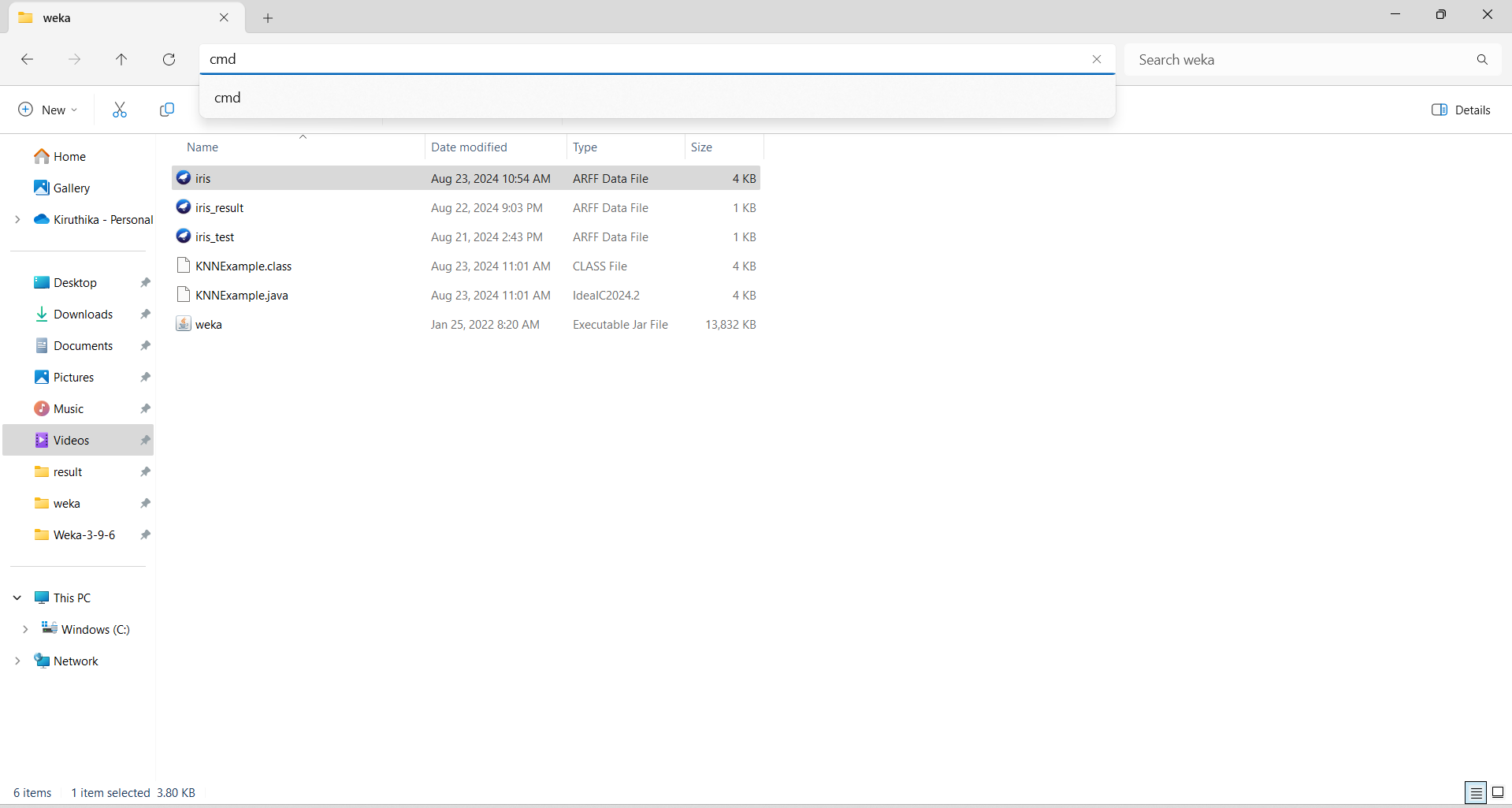
**In notepad:**

**Launch weka jar in your system**

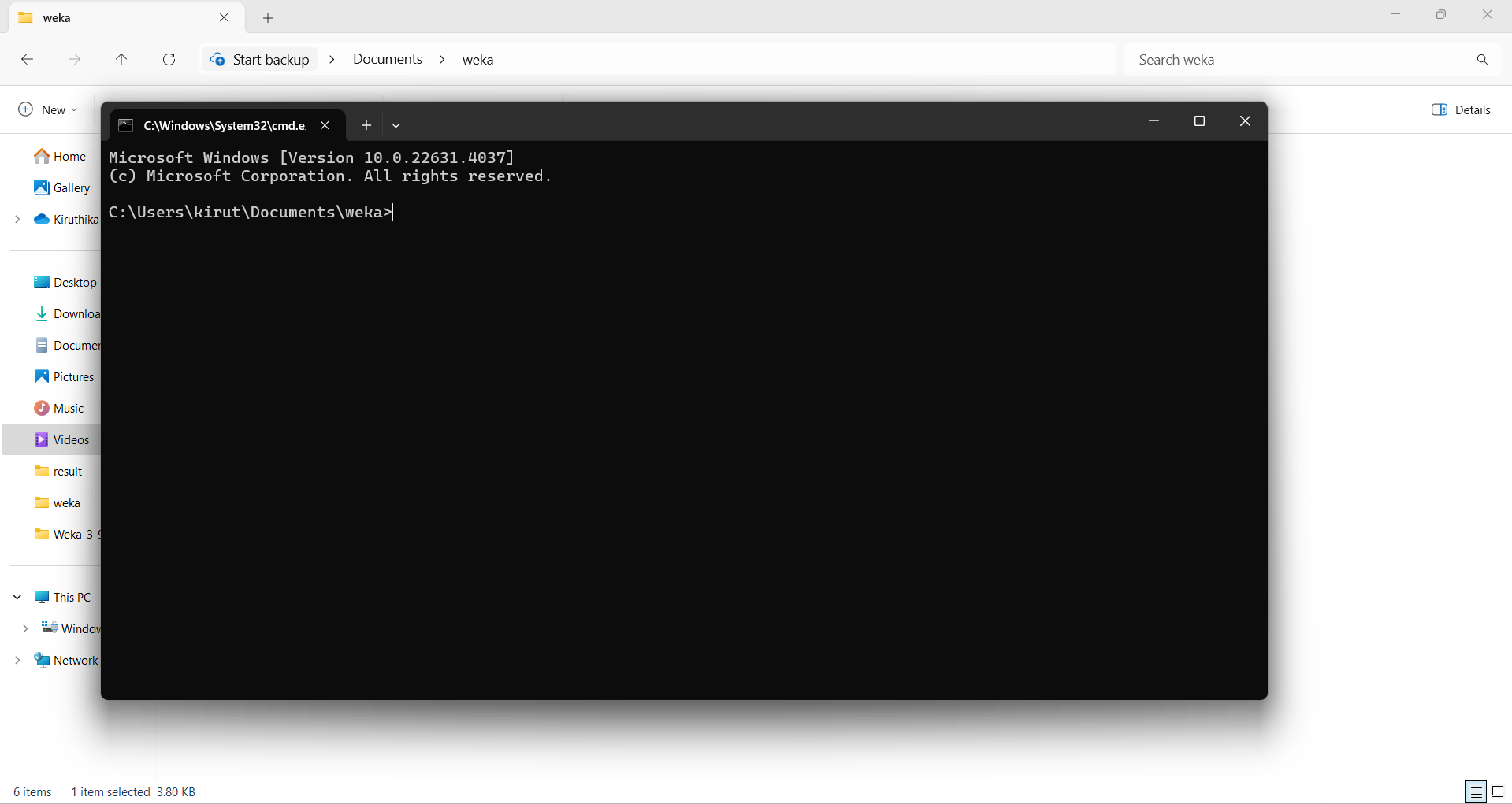
* Open notepad
* Write the program in the notepad
* Save the file where the datasets is stored and the weka is located



* Keep dataset ,weka and the program in one folder.
* Type command prompt in the location

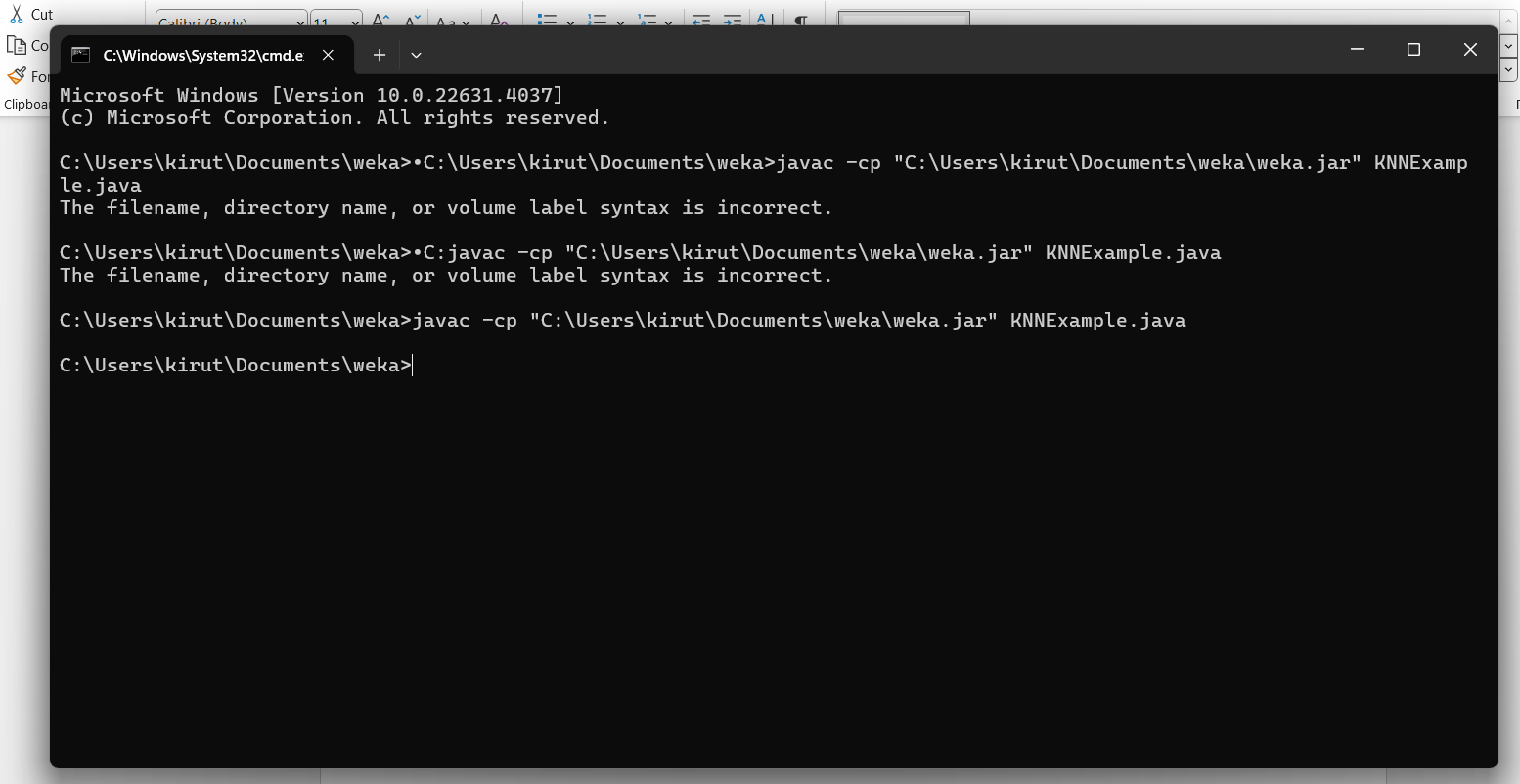


* Open command prompt.

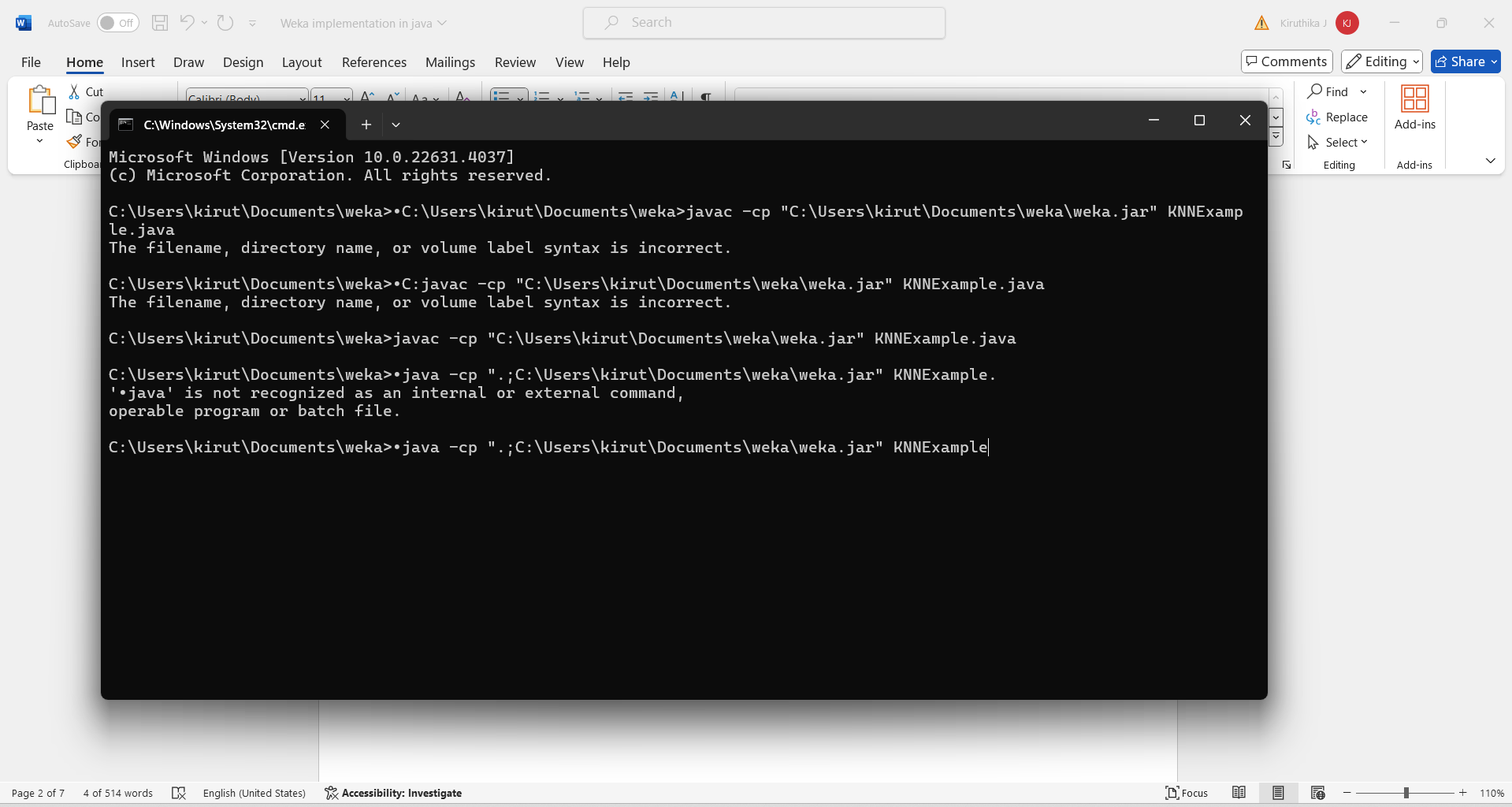


* Write command line to compile the program.
* Copy the path of weka jar file. Right click the weka jar file to copy the path.
* **Command for compile.**

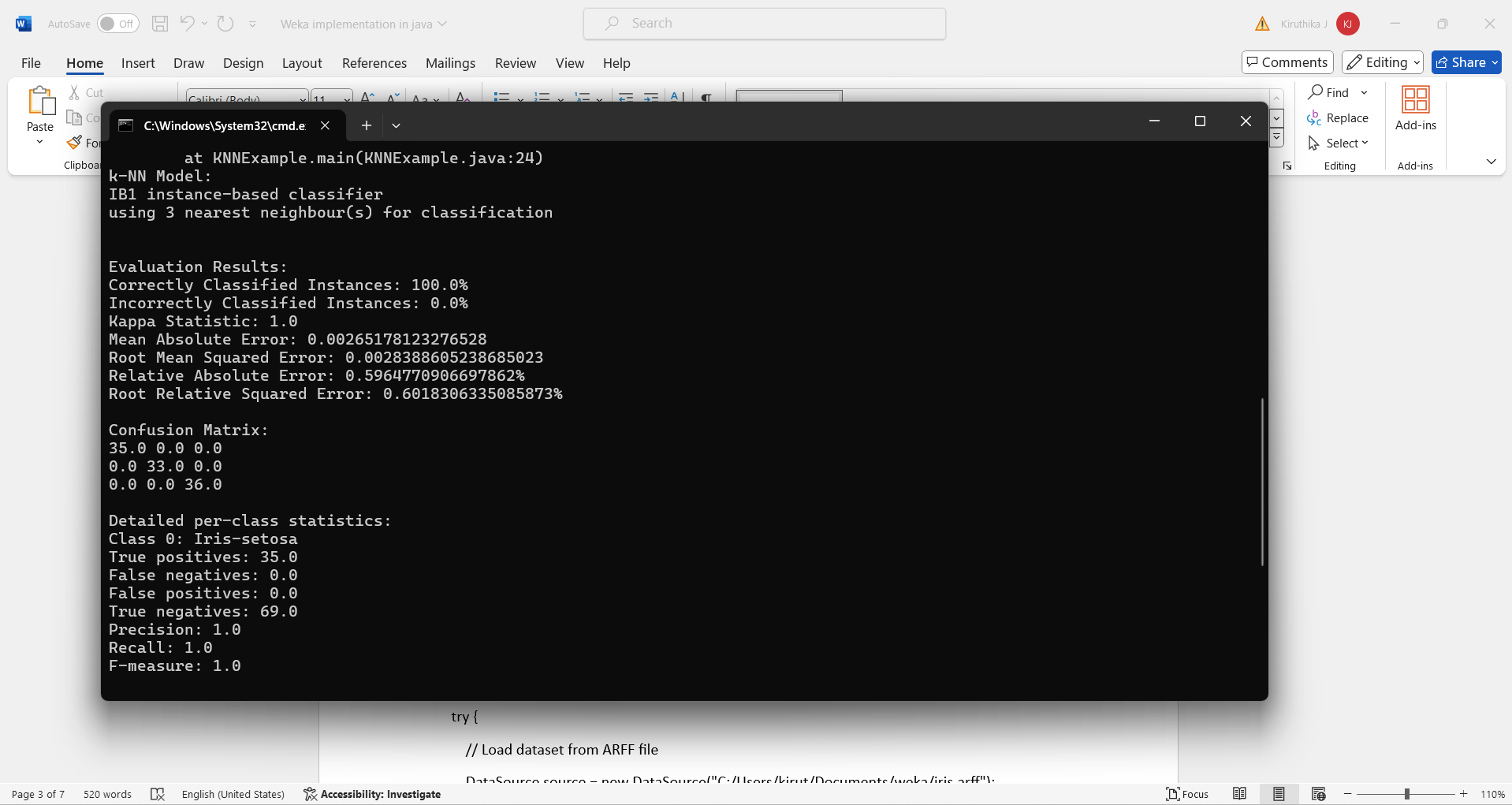
C:\Users\kirut\Documents\weka>javac -cp "C:\Users\kirut\Documents\weka\weka.jar" KNNExample.java



* **Command for run.**
* java -cp ".;C:\Users\kirut\Documents\weka\weka.jar" KNNExample.



Click enter to run the program.



**Program:**

import weka.classifiers.Classifier;

import weka.classifiers.lazy.IBk;

import weka.classifiers.evaluation.Evaluation;

import weka.core.Instances;

import weka.core.converters.ConverterUtils.DataSource;

public class KNNExample {

public static void main(String[] args) {

try {

// Load dataset from ARFF file

DataSource source = new DataSource("C:/Users/kirut/Documents/weka/iris.arff");

Instances data = source.getDataSet();

// Set class index (the attribute to predict)

if (data.classIndex() == -1) {

data.setClassIndex(data.numAttributes() - 1);

}

// Initialize the k-NN classifier with k=3

IBk knn = new IBk(3)

// Build the classifier

knn.buildClassifier(data);

// Print the k-NN model summary

System.out.println("k-NN Model:");

System.out.println(knn);

// Evaluate the model using 10-fold cross-validation

Evaluation evaluation = new Evaluation(data);

evaluation.crossValidateModel(knn, data, 10, data.getRandomNumberGenerator(1));

// Print evaluation results

System.out.println("\nEvaluation Results:");

System.out.println("Correctly Classified Instances: " + evaluation.pctCorrect() + "%");

System.out.println("Incorrectly Classified Instances: " + evaluation.pctIncorrect() + "%");

System.out.println("Kappa Statistic: " + evaluation.kappa());

System.out.println("Mean Absolute Error: " + evaluation.meanAbsoluteError());

System.out.println("Root Mean Squared Error: " + evaluation.rootMeanSquaredError());

System.out.println("Relative Absolute Error: " + evaluation.relativeAbsoluteError() + "%");

System.out.println("Root Relative Squared Error: " + evaluation.rootRelativeSquaredError() + "%")

// Print confusion matrix

System.out.println("\nConfusion Matrix:");

double[][] confusionMatrix = evaluation.confusionMatrix();

for (int i = 0; i < confusionMatrix.length; i++) {

for (int j = 0; j < confusionMatrix[i].length; j++) {

System.out.print(confusionMatrix[i][j] + " ");

}

System.out.println();

}

// Print detailed per-class statistics

System.out.println("\nDetailed per-class statistics:");

for (int i = 0; i < data.numClasses(); i++) {

System.out.println("Class " + i + ": " + data.classAttribute().value(i));

System.out.println("True positives: " + evaluation.numTruePositives(i));

System.out.println("False negatives: " + evaluation.numFalseNegatives(i));

System.out.println("False positives: " + evaluation.numFalsePositives(i));

System.out.println("True negatives: " + evaluation.numTrueNegatives(i));

System.out.println("Precision: " + evaluation.precision(i));

System.out.println("Recall: " + evaluation.recall(i));

System.out.println("F-measure: " + evaluation.fMeasure(i));

System.out.println();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**output:**

**C:\Users\kirut\Documents\weka>java -cp ".;C:\Users\kirut\Documents\weka\weka.jar" KNNExample**

k-NN Model:

IB1 instance-based classifier

using 3 nearest neighbour(s) for classification

Evaluation Results:

Correctly Classified Instances: 100.0%

Incorrectly Classified Instances: 0.0%

Kappa Statistic: 1.0

Mean Absolute Error: 0.00265178123276528

Root Mean Squared Error: 0.0028388605238685023

Relative Absolute Error: 0.5964770906697862%

Root Relative Squared Error: 0.6018306335085873%

Confusion Matrix:

35.0 0.0 0.0

0.0 33.0 0.0

0.0 0.0 36.0

Detailed per-class statistics:

Class 0: Iris-setosa

True positives: 35.0

False negatives: 0.0

False positives: 0.0

True negatives: 69.0

Precision: 1.0

Recall: 1.0

F-measure: 1.0

Class 1: Iris-versicolor

True positives: 33.0

False negatives: 0.0

False positives: 0.0

True negatives: 71.0

Precision: 1.0

Recall: 1.0

F-measure: 1.0

Class 2: Iris-virginica

True positives: 36.0

False negatives: 0.0

False positives: 0.0

True negatives: 68.0

Precision: 1.0

Recall: 1.0

F-measure: 1.0