Course: DVA262 — Machine Learning Concepts

Objective: The purpose of the Lab is to apply your understanding of the k-NN algorithm by implementing the k-NN algorithm from scratch in the C++ programming using the Lab skeleton.

Lab 1: The k-NN Algorithm Implementation

Execution and demonstration

This lab is executed by a group of two students or individually. The completed lab should be demonstrated to a lab assistant.

Useful aids

- The course literature
- Lecture slides and notes
- Eigen https://eigen.tuxfamily.org/dox/GettingStarted.html
- Google & Internet

Part 1: k-NN for Classification

You will implement the k-NN algorithm for the classification problem in this task.

Dataset: The classification problem will be based on the **Iris dataset,** which is already available in the project folder.

The Visual Interface: The main form (i.e., MainForm.h) calls all the class objects and functions that are already implemented in the skeleton. However, you may need to edit some parts of the code to make it consistent with your code to display the results correctly.

- *Loading and splitting the dataset*
  The code for loading and preparing the dataset is already implemented, so no additional code is required for this part. Use the following function in your code:

  - You can use the function loadAndPreprocessDataset() to prepare the dataset, which is located in the class name DataLoader.cpp under DataUtils folder.

- Use the splitDataset() function in the class name DataPreprocessor.cpp under the DataUtils folder to split the dataset into training and test sets.

Your Task: You will mainly work with two classes: KNNClassifier.cpp and SimilarityFunctions.cpp, located under the Classification and Utils folder, respectively.

- *Creating class member functions*

The KNNClassifier.cpp class may contain some or all of the following member functions:

- Constructor: initialises the member variables.
- fit() function: assigns training dataset and labels.
- predict() function: this is the main function that calculates the similarity between instances of test and training data sets and returns the classification prediction. *Note*: check the returns data type.

Task 1: In this Lab work you must complete the predict() function.

The SimilarityFunctions.cpp class contains several similarity functions bodies which are not implemented.

Task 2: You must complete eulideanDistance() function to compute the similarity between instances.

Task 3: In addition, you should at least implement one other distance function apart from the eulideanDistance() in the SimilarityFunctions.cpp class. Now, use that function for similarity measures in the predict() function. Does the algorithm perform the same when you use eulideanDistance()? Demonstrate and discuss the results with lab assistant.

- *Evaluating*

When you complete the implementation, it is time to evaluate your algorithm by measuring accuracy, precision, recall and F1 score. The code for these metrics is already done, so no additional implementation is required.

- You can use accuracy(), precision(), recall() and f1Score() functions, which are located in the class name Metrics.cpp under the Evaluation folder.

Task 4: What value of *k* have you used? Try different values for *k*. What is the largest and smallest value of *k* you can choose? Discuss your findings during the demonstration.

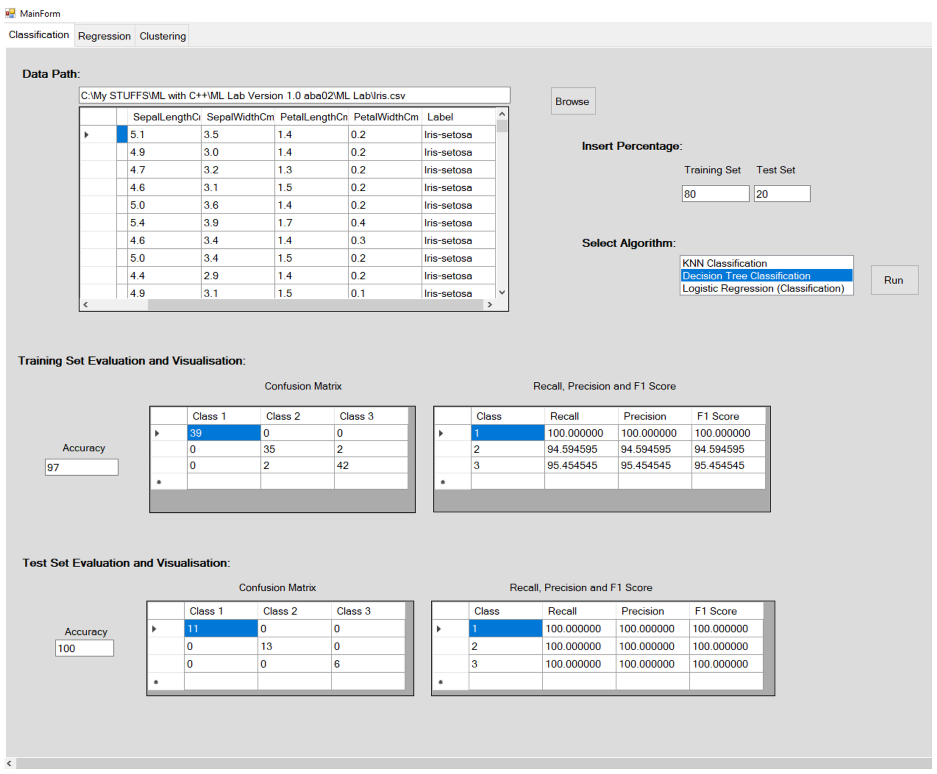
Note: The value of *k* is set when you create an object of the KNNClassifier.cpp class in the MainForm.cpp class.

- *Visualisation:*

A visual representation (for training and test data sets) is needed. The code for this task is already done, so use the following functions:

- Use the confusionMatrix() function located in the class name Metrics.cpp under the Evaluation folder, to calculate the confusion matrix for the specified number of class labels.
- Use the DisplayConfusionMatrixInGrid() function located in the MainForm.h, to Display the confusion matrix in the grid.

An example of the result from the k-NN classification should look shown in the Figure 1. The visual interface is same for all classifiers.



Data Path: C:\My STUFFS\ML with C++\ML Lab Version 1.0 aba02\ML Lab\Iris.csv

SepalLengthCı	SepalWidthCm	PetalLengthCn	PetalWidthCm	Label
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

Insert Percentage:

Training Set	Test Set
80	20

Select Algorithm:
- KNN Classification
- [✓ Decision Tree Classification]
- Logistic Regression (Classification)

Training Set Evaluation and Visualisation:

Accuracy: 97

Confusion Matrix

Class 1	Class 2	Class 3
39	0	0
0	35	2
0	2	42

Recall, Precision and F1 Score

Class	Recall	Precision	F1 Score
1	100.000000	100.000000	100.000000
2	94.594595	94.594595	94.594595
3	95.454545	95.454545	95.454545

Test Set Evaluation and Visualisation:

Accuracy: 100

Confusion Matrix

Class 1	Class 2	Class 3
11	0	0
0	13	0
0	0	6

Recall, Precision and F1 Score

Class	Recall	Precision	F1 Score
1	100.000000	100.000000	100.000000
2	100.000000	100.000000	100.000000
3	100.000000	100.000000	100.000000

Figure 1: Example result of Decision Tree classification shown in the visual interface. The k-NN algorithm uses the same visual interface.

Part 2: k-NN for Regression

In this task, you will implement the k-NN algorithm for the regression problem.

Dataset

The regression problem will be based on the **Boston Housing** dataset, which is already available in the project folder.

The Visual Interface: Similar to the classification task, the main form (i.e., MainForm.h) calls all the class objects and functions that are already implemented. But you may need to edit some parts of the code to make it consistent with your code to display the results correctly.

Your Task: You will mainly work with the KNNRegression.cpp class located under the Regression folder. If you have already completed the **Part 1** then you do not need to implement anything more in the SimilarityFunctions.cpp class.

- *Loading and splitting the dataset*

The code for loading and preparing (preprocessing) the dataset is already done, so no additional code is required for this part. Use the following function in your code:

- You can use the function readDatasetFromFilePath() to prepare the dataset, which is located in the class name DataLoader.cpp under DataUtils folder.
- Use the splitDataset() function in the class name DataPreprocessor.cpp under the DataUtils folder to split the dataset into training and test sets.

- *Creating class member functions*

The KNNRegression.cpp class will have the following member functions:

- Constructor: initialises the number of neighbours
- fit() function: assigns training dataset and labels.
- predict() function: this is the main function that calculates the similarity between instance of test and training data set and returns the prediction. *Note*: check the returns data type.

Task 5: In this part of the work you must complete the predict() function.

Task 6: If you have already completed **Task 2** and **Task 3,** you are good to go with the SimilarityFunctions.cpp class. Use the similarity function that you implemented in **Task 3** and show the result.

- *Evaluating*

Now, you need to evaluate the performance of the regression model measuring Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R-Squared. The code for these metrics is already implemented.

- You can use meanAbsoluteError(), rootMeanSquaredError(), and rSquared() functions, which are located in the class name Metrics under the Evaluation folder for this evaluation.

Task 7: What value of *k* have you used? Try different values for *k*. What is the largest and smallest value of *k* you can choose? Discuss your findings during the demonstration.

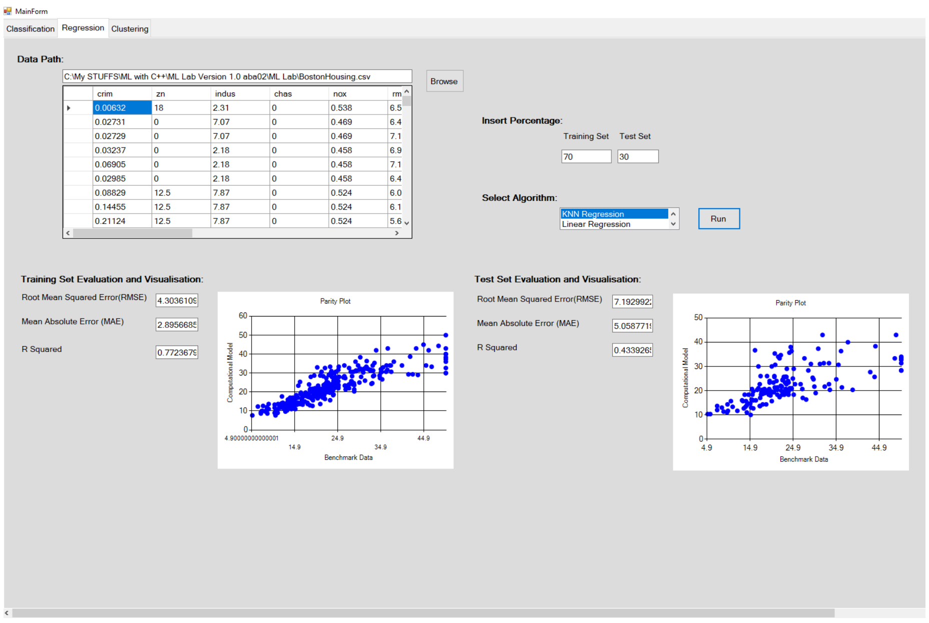
Note: The value of *k* is set when you create an object of the KNNRegression.cpp class in the MainForm.cpp class.

- *Visualisation*

A visual representation (for training and test sets) is needed. The code for this task is already done, so use the following functions for that:

- Use the VisualizeParityPlot() function, located in the MainForm.h, to display the parity plot for the actual and predicted test labels.

An example of the result from the k-NN regression is shown in the Figure 2. The visual interface is same for all regression algorithms.



Data Path:

C:\My STUFFS\ML with C++\ML Lab Version 1.0 aba02\ML Lab\BostonHousing.csv

crim	zn	indus	chas	nox	rm
0.00632	18	2.31	0	0.538	6.5
0.02731	0	7.07	0	0.469	6.4
0.02729	0	7.07	0	0.469	7.1
0.03237	0	2.18	0	0.458	6.9
0.06905	0	2.18	0	0.458	7.1
0.02985	0	2.18	0	0.458	6.4
0.08829	12.5	7.87	0	0.524	6.0
0.14455	12.5	7.87	0	0.524	6.1
0.21124	12.5	7.87	0	0.524	5.6

Insert Percentage:

Training Set: 70 Test Set: 30

Select Algorithm: KNN Regression / Linear Regression

Training Set Evaluation and Visualisation:

Root Mean Squared Error(RMSE)	4.3036109
Mean Absolute Error (MAE)	2.8956685
R Squared	0.7723679

Test Set Evaluation and Visualisation:

Root Mean Squared Error(RMSE)	7.192992
Mean Absolute Error (MAE)	5.058771
R Squared	0.433926

Figure 2: Example results of k-NN Regression in the visual interface.

Some Hints:

- Include the necessary header files at the beginning of your code.
- See lecture slides to predict labels considering k-nearest neighbours.
- Calculate the distance between the test data point and each training data point.
- Store predicted labels for all test data points.

Extra Assignment – if you want to learn some more.

Implement all the similarity functions in the SimilarityFunctions.cpp class. Test the performance of k-NN algorithm for both Classification and Regression using each similarity function in the predict() function. Which similarity function works best?