Project 2 Design Document

Chris Major

Tysen Radovich

Farshina Nazrul

Allen Simpson

September 26, 2019

Revision 1

Date	Description	Revision	Editor
09/26/2019	Created Document	0	Allen Simpson
09/27/2019	Added UML Diagram	1	Allen Simpson
09/27/2019	Added Introduction Section and References	2	Chris Major
09/27/2019	Added Design Overview	3	Tysen Radovich
09/27/2019	Added References	4	Farshina Nazrul

Contents

1. Introduction

1.1. Purpose

The purpose of this document is to describe the implementation of the Assignment 2 algorithms as described in the given project requirements. The algorithms implemented perform classifications and regression on six different sets of input data points, provided by the UCI Machine Learning repository (Dua and Graff (2017)).

1.2. Scope

This document describes the implementation details of the Project 2 algorithm. The algorithm will consist of a 5 major functions. First, is an implementation of K-Nearest Neighbor as described in class and according to our sources in both classification and regression form (Altman (1992)). Second is implementation K-Edited which will produce datasets that will be used in later algorithms (Donghai Guan et al. (2008)) . Third is an implementation of K-Condensed which will also produce datasets to be used in later algorithms (Hart (1968)). Fourth is an implementation of K-Means clustering (Hansen (2009)). Fifth is an implementation of K-Means Medoid (Swami and Jain (2006)). This document will specify the inputs and the testing of the algorithm.

2. Design Overview

2.1. Description of Problem

Given three classification data sets and three regression data sets, we are tasked with performing three different algorithms and two variations of one algorithm to either classify the data given or perform regression on a set. Our experiment's objective lies in determining the effects of various k values in the effectiveness of the classification and regression.

2.2. Technologies Used

The Project 2 algorithm will be programmed to run independently from the F# Interactive Window in Visual Studio. The development environment is Microsoft Visual Studio 2018. The algorithms will be programmed in the F# Functional Language.

2.3. Experimental Design

We will be using 0/1 Loss and Mean-Squared-Error to track loss. This will provide us an experimental value for how accurate our algorithm is.

3. Requirements Traceability

Requirement	Description	Design Reference
[r1.*]	Preprocessing	§4 Fig. 1
[r2.*]	KNearestNeighbor	§4 Fig. 3
[r3.*]	Edited-KNearestNeighbor	§4 Fig. 4
[r4.*]	Condensed-KNearestNeighbor	§4 Fig. 5
[r5.*]	KMeans	§4 Fig. 6
[r6.*]	KMeansMedoid	§4 Fig. 7

4. Driver Interface

Figure 1 depicts the method for preprocessing.

KNearestNeighbor (k:int) (trainingSet:IClassifiedPoint[])	
Input	The number of neighbors the algorithm will look at is requested along with the trainingSet that the algorithm is analyzing.
Output	Returns a classifier function that will take a point and return a string, which is the class of that point.
Description	Method returns the class of a point given a trainingSet and the number of neighbors

Figure 1: Preprocessing Method

Figure 2 depicts the UML Model for the Project 2 Algorithm.

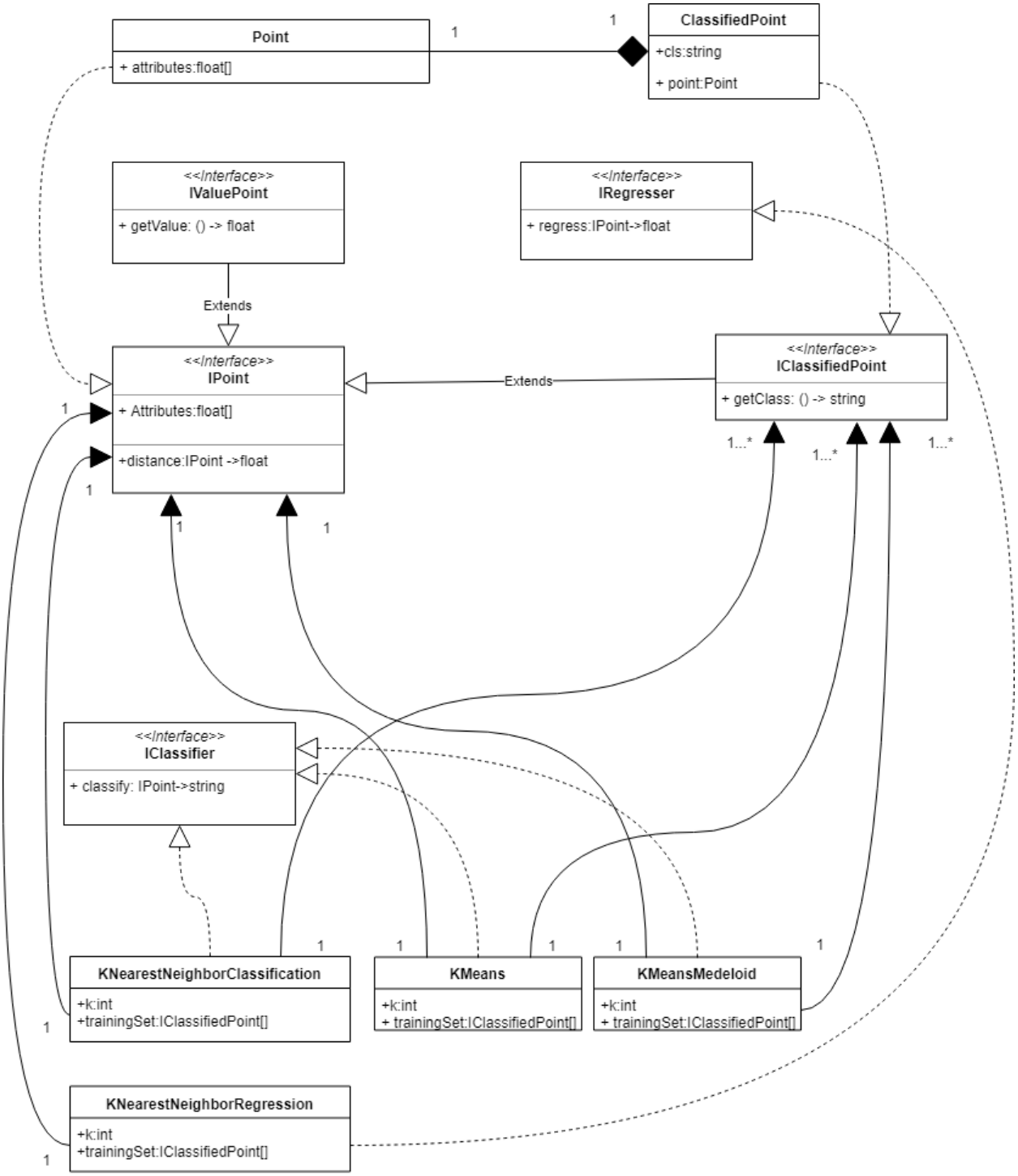

Figure 2: UML For Project 2 Algorithm

In Figure 2, each point is of a type of array of floats. A point implements an interface for IPoint. A point has only one classified point, but a classified point can exist without having a point in it. Therefore a composition was used instead of aggregation. Each classified point implements an interface of IClassifiedPoint where it also inherits from the IPoint. The IPoint interface uses each of these only once for KNearestNeighborClassification, KNearestNeighborRegression, KMeans,

KMeansMedoid. Each of these can only be implemented once because there does not need to be used on each of these points multiple times. This means that I do not need to use a single point multiple times for a single method. Each IClassifiedPoint can be used by multiple KNearestNeighborClassification, KNearestNeighborRegression, KMeans, KMeansMedoid classes.

5. Object Models

The Algorithm classes control the classification and regression of points. Each class's attributes work like the inputs to a function.

Figure 4 depicts the KNearestNeighbor process. Figure 5 depicts the process for Edited-KNearestNeighbor. Figure 6 depicts the process for Condensed-KNearestNeighbor. Figure 7 depicts the process for KMeans. Figure 8 depicts the process for KMeansMedoid.

KNearestNeighbor (k:int) (trainingSet:IClassifiedPoint[])	
Input	The number of neighbors the algorithm will look at is requested along with the trainingSet that the algorithm is analyzing.
Output	Returns a classifier or regressor function that will take a point and return a string, which is the class of that point.
Description	Method returns a classifier that will give the class of a point, given a trainingSet and the number of neighbors

Figure 3: KNearestNeighbor Method

Edited-KNearestNeighbor (startingSet:IClassifiedPoint[])	
Input	The set of classified points that will be trimmed down.
Output	Returns a reduced set of points.
Description	Method returns a subset of the points of startingSet by removing elements that are close and share classes.

Figure 4: Edited KNearestNeighbor Method

Condensed-KNearestNeighbor (startingSet:IClassifiedPoint[])	
Input	The set of classified points that will be trimmed down.
Output	Returns a reduced set of points.
Description	Method returns a subset of the points of startingSet by adding elements from a seed point that are close enough to present elements.

Figure 5: Condensed-KNearestNeighbor Method

KMeans (k:int) (trainingSet:IClassifiedPoint[])	
Input	The number of neighbors the algorithm will look at is requested along with the trainingSet that the algorithm is analyzing.
Output	Returns a classifier or regressor function that will take a point and return a string, which is the class of that point.
Description	Method returns a classifier that will give the class of a point, given a trainingSet and the number of neighbors using centroids

Figure 6: KMeans Method

KMeansMedoid (k:int) (trainingSet:IClassifiedPoint[])	
Input	The number of neighbors the algorithm will look at is requested along with the trainingSet that the algorithm is analyzing.
Output	Returns a classifier or regresser function that will take a point and return a string, which is the class of that point.
Description	Method returns a classifier that will give the class of a point, given a trainingSet and the number of neighbors using medoids

Figure 7: KMeansMedoid Method

6. References

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992

Donghai Guan, Weiwei Yuan, Young-Koo Lee, and Sungyoung Lee. Semi-supervised nearest neighbor editing. *In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence),* pages 1183–1187, June 2008.doi: 10.1109/IJCNN.2008.4633949.

P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968. doi: 10.1109/TIT.1968.1054155.

Grassi, Samuel Everett. "Building & Improving a K-Nearest Neighbors Algorithm in Python." *Medium*,Towards Data Science , 7 July 2018.Web. 27 Sept. 2019.

BE Hansen. Nearest neighbor methods. nonparametric econometrics. University of Wisconsin-Madison, 2009.

DK Swami and RC Jain. : Partitioning around medoids for classification. *Information Technology Journal*, 5(6):1102–1105, 2006