CSCI 447 Project 2 Design Document

George Engle, Troy Oster, Dana Parker, Henry Soule

September 27, 2019

1 Introduction

For this assignment, we are required to assess the performance of five different algorithms on six different data sets. The five algorithms used are k-nearest neighbors, edited k-nearest neighbors, condensed k-nearest neighbors, k-means clustering, and k-medoids clustering. The essential purpose of the latter four algorithms is to produce a reduced data set with which to run k-nearest neighbors. We have found literature suggesting improved accuracy among nearest neighbor algorithms that employ data reduction [1][2]. Based on this, we have hypothesized that the average accuracy of k-nearest neighbors across all six data sets without reduction will be worse than the average accuracy across the six data sets once reduction has been performed via edited k-nearest neighbors, condensed k-nearest neighbors, k-means clustering, and k-medoids clustering.

2 Class Descriptions

Path_Manager

This class handles pathing through the projects databases.This way it is easy to reconfigure or setup for a new database or file type at a moments notice.

Database

Our database class acts as an instantiable object for holding relevant information from a selected database (project data files).

Process_Data

Makes use of the Path_Manager to target a specified database, then loads in the relevant information from that database, then loads in the relevant information from that databases directory into a database object which can then be manipulated as we please.

-This class also contains functionality for identifying and correcting missing values from a database via bootstrapping statistic.

-It also contains functionality for randomly shuffling data in a given database by specific attribute and percentage to modify.

KNN: TODO: INCLUDE CLASS DESCRIPTION HERE?

Components TODO: REFACTOR INTO OTHER FORMAT!

Our design has 5 primary components. *database.py* is a wrapper class that will handle all functionality of each data set. Each instance of the database class will store the processed data of one our six data sets, the the index of the class attribute of its respective database. *knn.py* will store our implementations of k-nearest neighbors, edited k-nearest neighbors, and condensed k-nearest neighbors. *kcluster.py* will store the implementation of both clustering algorithms–k-means and k-medoids clustering. *validation.py* will perform our 10-fold cross-validation and our loss functions. *main.py* will perform the execution of our program.

3 Design Decisions

Missing Data handling

When a dataset is identified to have missing data, we identify the sets of data with missing attributes, and move them from the databases stored data matrix into a separate list of the sets of data with missing attributes. Once the missing data has been identified, we substitute values for the missing attribute via bootstrapping random values from the set of data that does not contain any missing attributes as our means of imputation.

We opted not to simply remove data, even if under a certain threshold, as we assumed it would be more likely to impact our results than including data that would be discredited if incorrect by other present data. Additionally, we dont have a static value to symbolize a missing value, but rather one must be specified for a given dataset in the database directorys .attr file. This approach allows missing values to be considered a category of their own if necessary.

Identifying Categorical vs Discrete (continuous) data

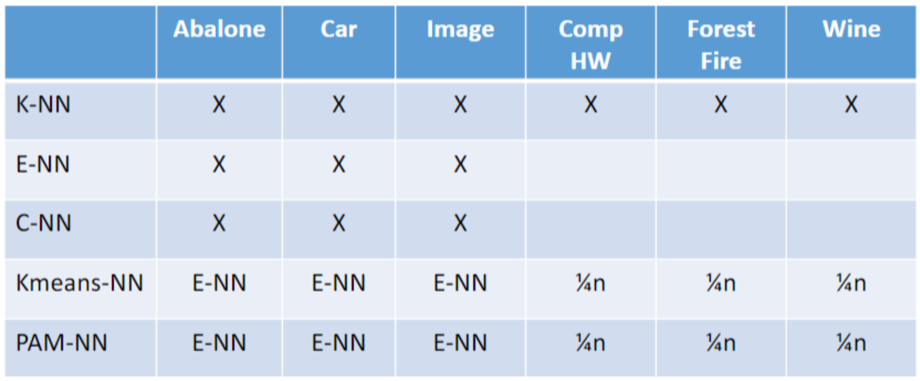
In order to determine how we should process data, we have decided to use our method from assignment 1 for identifying categorical vs discrete data types to automatically handle which algorithms will be available/valid to use on a given data set.

Nearest Neighbors distance function

We decided on Euclidean distance as our means of determining nearest neighbors, as manhattan distance didnt seem to make much sense seeing as it isnt direct.

4 Experimental Design

In order to test our implementations of the algorithms, we will use classification and regression datasets as indicated in Figure A below.



	Abalone	Car	Image	Comp HW	Forest Fire	Wine
K-NN	X	X	X	X	X	X
E-NN	X	X	X			
C-NN	X	X	X			
Kmeans-NN	E-NN	E-NN	E-NN	¼n	¼n	¼n
PAM-NN	E-NN	E-NN	E-NN	¼n	¼n	¼n

Figure A: Project 2 Clarification chart

For each test, we will use ten-fold cross validation with the equal width binning approach for setting up the training and testing bins from the given dataset.

Handling Tuning

We will be following the rule of thumb to use the square root of the number of training samples as the initial value for the amount of neighbors (K) used for the nearest neighbors algorithms. From here, however we want to see if we can improve our ability to more accurately represent the model by increasing and decreasing the value of K and checking if it is a better fit.

It is worth noting however, that this method is susceptible to the hill climbing problem, however working our way around that with more iterations would likely result in a drastic increase in the amount of time it takes to find the supposed optimal K value, so we may not end up with a perfect solution as we will likely forgo this additional step.

Evaluation

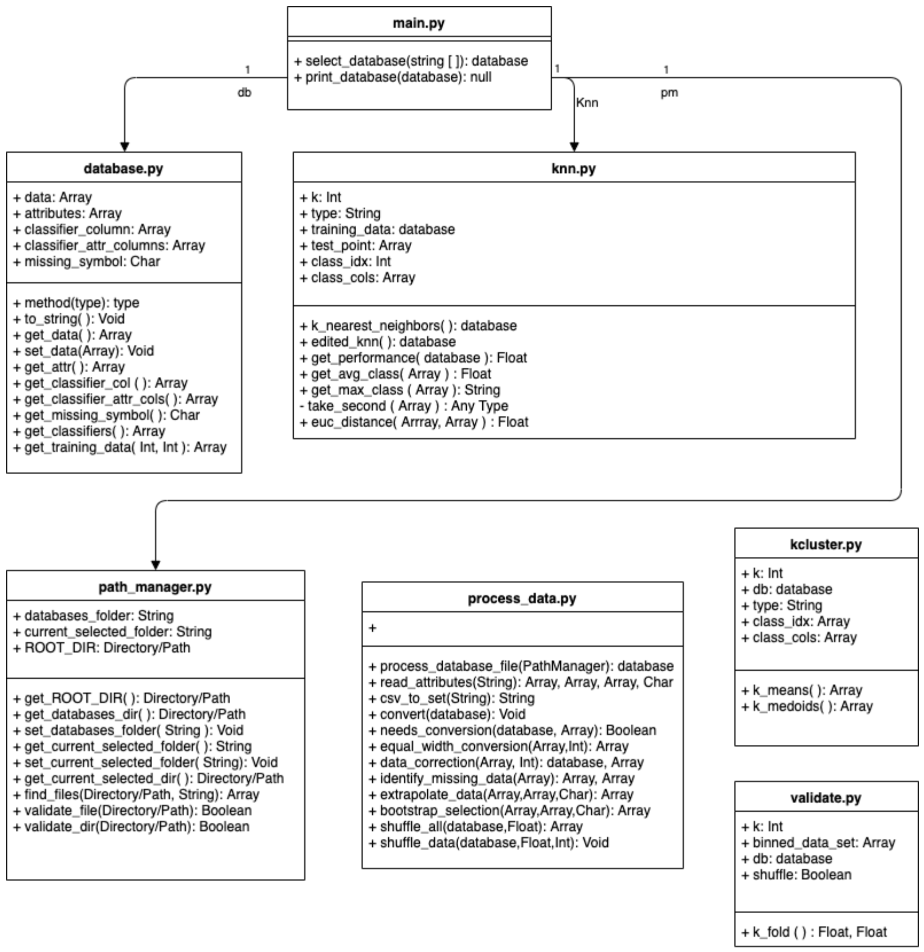
In order to properly evaluate our implementations, we will split up our evaluation methods for regression and classification based problems into separate functions. This is necessary seeing as classification doesn't involve real values, whereas the regression ones do, hence the innate disconnect and reasoning for identifying and evaluating them separately.

For evaluating regression problems, we need a way of comparing our methods predicted continuous value to the actual continuous value. But we also need this method of measuring our prediction accuracy to be robust with regards to outliers.

So we use Mean Absolute Error (MAE), which measures the sum of the absolute differences between our predicted continuous values and the actual continuous values, and then averages the value by the amount of values compared. ($MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$) As a bonus, it averages the absolute difference is important to note, as it makes MAE resistant to outliers by weighting the individual differences equally. The closer MAE is to 0, the better our predicted line of best fit.

As for evaluating how well our classification predicting implementation performs, we need a way of comparing our predicted classifications and the actual classifications. But this time we dont really care about outliers, so we use the Mean Squared Error as it tells us how far off our prediction distribution is from the actual distribution (The smaller the MSE, the better our ability to accurately predict the correct classification), and accuracy as metrics for algorithm evaluation. We use accuracy in addition because Mean Squared Error on its own can possibly be 0 on its own which would lead to incorrect classifications.

UML Diagram


main.py
+ select_database(string [ ]): database
+ print_database(database): null
1
db
1
Knn
1
pm
database.py
+ data: Array
+ attributes: Array
+ classifier_column: Array
+ classifier_attr_columns: Array
+ missing_symbol: Char
+ method(type): type
+ to_string( ): Void
+ get_data( ): Array
+ set_data(Array): Void
+ get_attr( ): Array
+ get_classifier_col ( ): Array
+ get_classifier_attr_cols( ): Array
+ get_missing_symbol( ): Char
+ get_classifiers( ): Array
+ get_training_data(Int, Int): Array
knn.py
+ k: Int
+ type: String
+ training_data: database
+ test_point: Array
+ class_idx: Int
+ class_cols: Array
+ k_nearest_neighbors( ): database
+ edited_knn( ): database
+ get_performance(database): Float
+ get_avg_class(Array) : Float
+ get_max_class (Array): String
- take_second (Array) : Any Type
+ euc_distance(Arrray, Array) : Float
path_manager.py
+ databases_folder: String
+ current_selected_folder: String
+ ROOT_DIR: Directory/Path
+ get_ROOT_DIR( ): Directory/Path
+ get_databases_dir( ): Directory/Path
+ set_databases_folder(String): Void
+ get_current_selected_folder( ): String
+ set_current_selected_folder(String): Void
+ get_current_selected_dir( ): Directory/Path
+ find_files(Directory/Path, String): Array
+ validate_file(Directory/Path): Boolean
+ validate_dir(Directory/Path): Boolean
process_data.py
+
+ process_database_file(PathManager): database
+ read_attributes(String): Array, Array, Array, Char
+ csv_to_set(String): String
+ convert(database): Void
+ needs_conversion(database, Array): Boolean
+ equal_width_conversion(Array,Int): Array
+ data_correction(Array, Int): database, Array
+ identify_missing_data(Array): Array, Array
+ extrapolate_data(Array,Array,Char): Array
+ bootstrap_selection(Array,Array,Char): Array
+ shuffle_all(database,Float): Array
+ shuffle_data(database,Float,Int): Void
kcluster.py
+ k: Int
+ db: database
+ type: String
+ class_idx: Array
+ class_cols: Array
+ k_means( ): Array
+ k_medoids( ): Array
validate.py
+ k: Int
+ binned_data_set: Array
+ db: database
+ shuffle: Boolean
+ k_fold ( ) : Float, Float

5 Works Cited

[1] Wagner, T. "Convergence of the Edited Nearest Neighbor (Corresp.)." IEEE Transactions on Information Theory, vol. 19, no. 5, 1973, pp. 696?697., doi:10.1109/tit.1973.1055059.

[2] Nitin Bhatia, V. "Survey of Nearest Neighbor Techniques.: International Journal of Computer Science and Information Security, vol. 8, no. 2. 2010.