Team 36 Project 2 Design Document

Max Hymer HYMERMAX1@GMAIL.COM
Department of Electrical and Computer Engineering
Montana State University
Bozeman, MT 59715, USA

Michael Downs MICHAELD20312@GMAIL.COM
Division of Computer Science
Montana State University
Bozeman, MT 59715, USA

Date: September 25, 2024

1 System Requirements

1.1 Requirement One

: Download the six (6) data sets from the UCI Machine Learning repository. You can find this repository at http://archive.ics.uci.edu/ml/. The data sets are also available in Brightspace. This requirement is very straightforward - we will simply download the breast cancer, glass, soybean (small), abalone, computer hardware and forest fire .data and .name files from Brightspace. This is an important step in analyzing how the datasets are oriented so that we can fit our pre-processing algorithm to the data.

1.2 Requirement Two:

Pre-process the data to ensure you are working with complete examples (i.e., no missing attribute values). We intend to read in the data then replace all instances of "?" (as that is what marks an incomplete data point in the relevant data set (breast cancer)) in the data with a random value within the constraints of the feature (i.e. if the feature contains integers from one to ten, we replace any "?" within that feature with a random integer from 1-10). This is a simple yet effective format of data imputation that produced good results for us with the Naive Bayes classifier, so we decided to use it again for this project.

1.3 Requirement Three:

If appropriate, apply a suitable method for dealing with categorical features. We intend to deal with categorical features using one-hot coding. We feel that the simplicity of this approach is a strength; we are less likely to create errors or accidentally confound our data with this approach than we would be with a more complex approach. So, we will likely spend less time troubleshooting our preprocessing with this approach. Basically, we feel that this approach is best because it will allow us to focus on the algorithm and hyperparameter tuning rather than prepossessing; leading to greater learning and understanding of the algorithm its hyperparameter tuning. This method adds a layer to the experiment as well; it allows us to compare the algorithm's performance on relatively unaltered categorical data to its performance on unaltered continuous data.

1.4 Requirement Four:

Implement k-nearest neighbor and be prepared to find the best k value for your experiments. You must tune k using the approach described above. We intend to implement the k-nearest neighbor algorithm in a single class that handles both categorical and regression data sets. For categorical data sets, our algorithm will get the majority (plurality) vote from k nearest data points to determine the category of the unknown point. For regression data sets, our algorithm will apply a Gaussian kernel to weight each of the k nearest data points based on their distance and will calculate a weighted average for the regression value of the unknown point based on these weights. We will tune k by extracting 10% of the (stratified) data (at random) to be used for tuning. For our training set, we will test against this 10% with different parameter values and pick the best model. Then, we will apply the model that goes with those tuned values against your test set. To be specific, since we are doing 10-fold cross-validation, we first take out 10% for tuning. Then, from the remaining 90%, split into ten folds of 9% of the data each. For each of the experiments, we will combine nine of the folds, holding out the tenth as your test set. To tune, we will train 10 times, once on each of the nine fold sets while testing with the initial 10%. We will average the result and take the best hyperparameters based on those averaged results. Then, we will evaluate generalization ability on the held-out fold.

1.5 Requirement Five:

Implement edited k-nearest neighbor. See above with respect to tuning k. On the regression problems, you should define an error threshold ϵ to determine if a prediction is correct or not. This ϵ will need to be tuned. We intend to define an error threshold ϵ based on the context of each data set. We will use the range and variance of each data set along with our knowledge of the domain of the data set to pick an initial error threshold ϵ. Then, we will tune ϵ by first running our data through 10-fold cross validation and evaluating both the average F1 and 0/1 loss scores then changing ϵ based on these values. We intend to implement edited k-nearest neighbor using k=1. To implement edited k-nearest neighbor, we plan to first take one point out of the training set and run k-nearest neighbor on the remaining points. Then, we will check if the point's label or regression value (using our error threshold) agrees with the majority of its k_n neighbors. Finally, we will remove points that do not agree with the majority of their k_n neighbors. The remaining points after this process is completed become the edited k-nearest neighbor training data set.

1.6 Requirement Six:

Implement k-means clustering and use the cluster centroids as a reduced data set for k-NN. For this, note the the number of clusters is different from the number of neighbors. It might be better think of the former as k_c and the latter as k_n to keep them straight. To avoid tuning, you should choose a number of clusters approximately equal to the number of examples returned from edited nearest neighbor. We plan to first perform k-means clustering. To do this, we first randomly initialize k_c centroids (with the number of centroids equal to the number of points in the edited k-nearest neighbor training data set). Then, we will assign each data point to the nearest centroid. Lastly, we will update the position of the centroid by calculating the mean of all points assigned to it. We will repeat this assignment and averaging process until the positions of the centroids don't change significantly. We will use the final centroids as the training data set for k-nearest neighbor. This way, the training data set for k-nearest neighbor is much smaller than the unaltered data set would have been. However, it is still accurately representative of the unaltered data set, so it is simply a more efficient but more complex way to perform the k-nearest neighbor algorithm on our data sets. So, we will be performing it in this fashion on our data sets.

1.7 Requirement Seven:

For classification, employ a plurality vote to determine the class. For regression, apply a Gaussian (radial basis function) kernel to make your prediction among the neighbors. You will need to tune the bandwidth σ for the Gaussian kernel. For categorical data sets, our algorithm will get the majority (plurality) vote from k nearest data points to determine the category of the unknown point. Basically, the most common class among the k nearest data points is the class that will be assigned to the unknown point. For regression data sets, our algorithm will apply a Gaussian kernel to weight each of the k nearest data points based on their distance and will calculate a weighted average for the regression value of the unknown point based on these weights. We plan to use the Gaussian kernel formula

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$$

where x is the query point, x_i is the i^{th} neighbor, and σ is the bandwidth parameter of the Gaussian kernel that controls the width of the kernel to do this. First, we plan to use Euclidean distance to find the k nearest neighbors. Then, we apply the Gaussian kernel

$$w_i = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$$

to calculate the weight of each neighbor. Lastly, we plan to calculate the weighted sum of the neighbors' values

$$\hat{y} = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

where y_i is the value of the i^{th} neighbor and w_i is the corresponding weight. We plan to tune σ by first running our data through 10-fold cross validation and evaluating both the average F1 and 0/1 loss scores then changing σ based on these values.

1.8 Requirement Eight:

Develop a hypothesis focusing on final performance of each of the chosen algorithms for each of the various problems. We plan to develop hypotheses based on our knowledge of each data set along with our knowledge of how each algorithm functions. With this information, we feel we can predict with at least a reasonable degree of accuracy the final performance of each of the chosen algorithms on each of the data sets.

different hypothesis... maybe another?

We want to predict that data that corresponds well with higher k values and data that corresponds well with a lower k value will likely be in the same database and never allow us to find perfect accuracy. This is where our edited class should perform much better as the information is more uniform, represented well by a specific k value throughout all class relationships.

1.9 Requirement Nine:

Write a final paper encompassing the results of our tests. This will be an extremely similar document to this one, with all of our testing, reflection on hypothesis and analysis of our data structuring, flow and test strategy.

1.10 Requirement Ten:

Make a video demonstrating the results of our code. This video will focus on the behavior of our code, will show input, data structure, and output, will be less than five minutes in length, will be posted on Microsoft Stream, will only employ fast-forwarding through long computational cycles, and will contain verbal commentary explaining the elements we are demonstrating. As far as project 2 specific requirements go, it will show our data being split into ten folds for one of the data sets, the calculation of our distance function, the calculation of our RBF kernel function, an example of a point being classified using k-nn (including the neighbors returned as well as the point being classified and its prediction), an example of a point being regressed using k-nn (including the neighbors returned as well as the point being predicted and its prediction), an example being edited out of the training set using edited nearest neighbor, a data point being associated with a cluster while performing k-means clustering, the average performance across the ten folds for each of k-nn, ENN, and k-means k-NN on a classification data set, and the average performance across the ten folds for each of k-nn, ENN, and k-means k-NN on a regression data set.

2 System Architecture

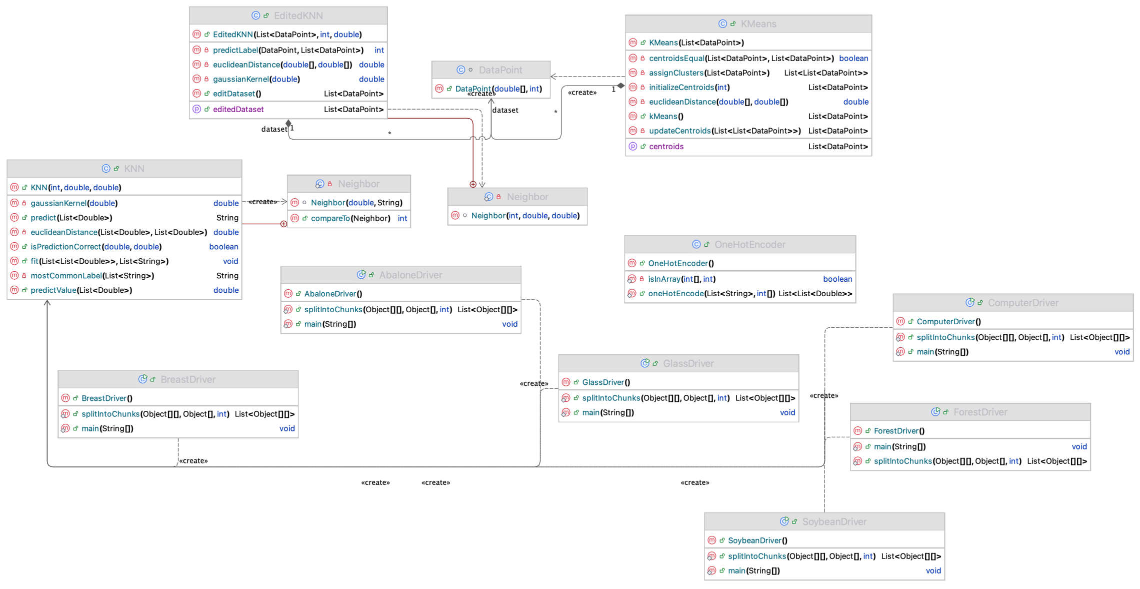
EditedKNN
EditedKNN(List<DataPoint>, int, double)
predictLabel(DataPoint, List<DataPoint>) int
euclideanDistance(double[], double[]) double
gaussianKernel(double) double
editDataset() List<DataPoint>
editedDataset List<DataPoint>
DataPoint
DataPoint(double[], int)
KMeans
KMeans(List<DataPoint>)
centroidsEqual(List<DataPoint>, List<DataPoint>) boolean
assignClusters(List<DataPoint>) List<List<DataPoint>>
initializeCentroids(int) List<DataPoint>
euclideanDistance(double[], double[]) double
kMeans() List<DataPoint>
updateCentroids(List<List<DataPoint>>) List<DataPoint>
centroids List<DataPoint>
dataset
1
*
«create»
KNN
KNN(int, double, double)
gaussianKernel(double) double
predict(List<Double>) String
euclideanDistance(List<Double>, List<Double>) double
isPredictionCorrect(double, double) boolean
fit(List<List<Double>>, List<String>) void
mostCommonLabel(List<String>) String
predictValue(List<Double>) double
Neighbor
Neighbor(double, String)
compareTo(Neighbor) int
Neighbor
Neighbor(int, double, double)
OneHotEncoder
OneHotEncoder()
isInArray(int[], int) boolean
oneHotEncode(List<String>, int[]) List<List<Double>>
AbaloneDriver
AbaloneDriver()
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
main(String[]) void
ComputerDriver
ComputerDriver()
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
main(String[]) void
GlassDriver
GlassDriver()
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
main(String[]) void
BreastDriver
BreastDriver()
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
main(String[]) void
ForestDriver
ForestDriver()
main(String[]) void
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
SoybeanDriver
SoybeanDriver()
splitIntoChunks(Object[][], Object[], int) List<Object[][]>
main(String[]) void

Figure 1: Project 2 UML Diagram

2.1 Drivers

Our driver classes are represented in the lower half of Figure 1. We created a unique driver for each data set that reads in the corresponding data set, performs unique preprocessing based on the needs of that data set (which includes calling the one-hot encode method if needed), performs 10-fold cross validation on the data set (the splitIntoChunks method is used to split the data set into 10 folds), calls the KNN (and will call EditedKNN or KMeans as well when we get to that stage in testing) for each fold, calculates and prints the accuracy, 0/1 loss, precision, recall, and F1 scores for each fold, and calculates and prints the average across all folds for these loss functions.

2.2 One-Hot Encoder

Our one-hot encoder class is designed to one-hot encode any categorical data present in the required data sets. It takes the initially preprocessed data and and an array containing the indices of the categorical columns within the array as arguments (so it knows which columns to one-hot encode) and returns a list containing both the newly one-hot encoded data and the already continuous data.

2.3 K-Nearest Neighbor Algorithm

Our k-nearest neighbor class performs the k-nearest neighbor algorithm. It is designed to work for both categorical and regression data sets. To do this, it takes in a k value, bandwidth, and error threshold in its constructor. The bandwidth and error threshold are not used on categorical data sets (so they can be set to any value) but are still required in the constructor for the regression implementation to work in the same class. Once the class is instantiated, a driver must call the fit method and provide it with the training data and labels as arguments. This method is similar to a constructor; it simply puts this data into the k-nearest neighbor object. For categorical data sets, the predict method is the method that actually performs the k-nearest neighbor algorithm using by using a Euclidean distance helper method and a plurality vote (done in the mostCommonLabel helper method) to actually determine a test instance's class. For regression data sets, the predictValue method performs the k-nearest neighbor algorithm by using a Euclidean distance helper method and a Gaussian kernel (done in the gaussianKernel helper method) to make its prediction. Then, the isPredictionCorrect method can be called with the prediction and actual value as arguments to check whether the prediction is within the defined error threshold. This class contains a private nested neighbor class as well. It is included to store the distance and label of each data point fed into the algorithm and overrides the compareTo method so that it compares the distance between two neighbors. Basically, this nested class exists to store information necessary to the k-nearest neighbor class in a way that is simple and easy to deal with.

2.4 Data Point

Our data point class is used to store individual data points within data sets. This way, each data point's features and labels are stored within a clearly defined object and are tied together.

2.5 Edited K-Nearest Neighbor

Our edited k-nearest neighbor class is designed to edit the training data sets to remove points that do not agree with the majority of their neighbors. It takes the training data set, the number of neighbors, and the desired bandwidth as arguments. It contains the same euclideanDistance and gaussianKernel helper methods found in the KNN class and the nested neighbor class for largely the same functions they were used for in that class. The editDataset method uses these helper methods as well as a predictLabel helper method to determine whether each point agrees with the majority of its neighbors and returns an updated training data set with only the points that agree with the majority of their neighbors.

2.6 K-Means Clustering

Our k-means class is designed to create centroids that represent our data set. It takes in the training data set as an argument. It contains the euclideanDistance helper method just like in the KNN class as well as assignClusters, initializeCentroids, updateCentroids, centroidsEqual helper methods that assist the kMeans method in assigning clusters to create and fit centroids that accurately represent the training data set. It returns a training data set made up of centroid data points rather than the original data points.

3 System Flow

In preprocessing, we will be taking all of our datasets in our program and creating representations of them depending on how they may need to be changed to all be processed by the same algorithm. For low occurrences in unknowns, ignoring them, or removing them is recommended. For our project, we plan on taking some random information within the bounds of the feature and implementing it at the unknown position. Thus, creating noise that may already be present and widespread within the data, in turn, creating our algorithm based on this fact and creating a program that may be more resilient to noise. Then, we will be using stratified 10-fold cross-validation to test our data in a manner that represents the dataset as a whole within each fold. To describe each fold in well respect to the general database, we plan on implementing a close ratio of classes in terms of the whole dataset within a similar ratio in each fold. Next, we will be passing each of these folds to our edited, clustered, and original k nearest neighbor algorithms. Our algorithms are going to train on most of the folds and test on one fold. Our k nearest neighbor plurality vote and Gaussian kernel algorithms will be returning information about the performance of each test fold. This information will include all of our data with our guesses and answers as well as our guess accuracy and F_1 score. After evaluating our loss functions, reflection and tuning will take place, but this is the general flow of the data through our algorithm.

4 Test Strategy

Our test strategy consists of running each 10-fold representation five times so that we have 50 total tests. This should be enough implementation to give us a general idea of the performance of our program. In terms of strategies, we intend on using for tuning our k value in our nearest neighbor algorithm. We found that some datasets contain past usages of k values, but we do not want to rely on this information. Instead, we are going to test this value and four other values. Our other values will be chosen as something much smaller, much larger, and a couple that are fairly similar. For data sets that contain no information about past k-value usage and results, we plan to test more. Once we have found a couple of our best values, we are going to try implementing k values between the two best found or just outside them, to see if better results are present.

5 Task Assignments/Schedule

$\longrightarrow$ Max Hymer
- Driver implementation (requirements 1 and 2) - 9/30
- One-hot coding implementation (requirement 3) - 9/30
- Cross validation implementation (requirement 4) - 9/30
- Develop hypotheses (requirement 8) - 10/2

$\longrightarrow$ Michael Downs
- K-nearest neighbor (KNN) implementation (requirements 4 and 7) - 9/30
- Edited and k-means KNN implementations (requirements 5 and 6) - 9/30
- Implement loss functions (requirement 4) - 9/30
- Edit code to show correct printouts, etc. for the video (requirement 10) 10/6

$\longrightarrow$ Working together
- Testing and tuning (requirements 4, 5, 6, 7) - 10/4
- Commenting and recording video presentation (requirement 10) - 10/7
- Final paper (requirement 9) - 10/7