Project 2

Team 14
Josh Aney - josh.aney@icloud.com
Owen Cool - ohcool57@gmail.com
Nicolas Perl - nicolas.leon.perl@gmail.com
25 September 2024

Requirements

For this project, it is required to download six datasets from the UCI Machine learning repository. Once the data sets are downloaded it will be required to do some data preprocessing. For almost all our datasets, no attribute values are missing, and for those that are missing attribute values, relatively few instances are missing values. Thus, to ensure all instances in the dataset have complete values, we will **remove all instances with missing attribute values**. The next phase of preprocessing will be for the categorical features, which will be dealt with by applying **one-hot encoding**. If performance is not satisfactory, other methods will be investigated.

It is also required to implement the **k-nearest neighbor algorithm**: Given an observed feature vector x, we use the k nearest neighbors of x among $x_1, \ldots, x_n$ and take a majority vote of the labels. The distance to the nearest neighbors will be calculated by using Minkowski Metrics with a p value of 0, 1, 2, or infinity.

$$D_p(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$$

Corresponding to these *k* nearest neighbors. For this algorithm, the *k* value for our experiments is tuned using the approach described below.

The **edited k-nearest neighbor algorithm** should also be implemented (Error removal version):

1. Each data point in the training set is classified using all the other training data points. However, if a chosen point is misclassified, it is removed from the training set.
2. Point 1 is iterated until the performance of the algorithm decreases.

On the regression problems, an error threshold **ϵ** is defined to determine if a prediction is correct or not, which will need to be tuned as well.

Additionally, **k-means clustering** is required to be implemented:

1. Choose k and initialize cluster centroids randomly
2. Calculate distances to all centroids and assign each data point its closest cluster centroid
3. Recompute centroids by computing the mean of each cluster
4. Stop after convergence (when mean does not change (“much”) anymore)

To avoid tuning, the number of clusters should be approximately equal to the number of examples returned by edited NN.

We use **ten-fold cross-validation** to train on nine of the partitions and test on the remaining partition. After rotating 10 times through the partitions and every partition was used exactly once as a test set, an average of the performance is computed. We will compare these results using the metrics of **precision, recall, accuracy, and F1-score**. Comparing these results, we will be able to **evaluate our hypothesis**.

For the regression data sets, the data is sorted based on the response target value in the data. The sorted data is then broken into groups of ten consecutive examples. Using these ten datasets, the first item from each group is drawn for the first data set, the second item for the second data set, and so on.
For **classification**, we will implement a **plurality vote** to determine the class. For **regression**, a **Radial Basis Function kernel** is applied to make our prediction among the neighbors, whose bandwidth, σ, will be tuned.

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

We will then formulate our **hypothesis**, predicting which dataset will see better performance using the k-NN algorithm, and which hyperparameter values will be most successful.

Hyperparameter tuning is done by extracting 10% of the data (random, but stratified). Then the remaining 90% are split into ten folds of 9%. For each of the experiments, nine of the folds are combined, holding out the tenth as our test set. Tuning is done by training 10 times, once on each of the nine fold sets while tuning with the initial 10%. The hyperparameters are chosen by selecting the best of the averaged results. The **generalization ability** is tested 10 times, one for each unique held-out fold.

System Architecture

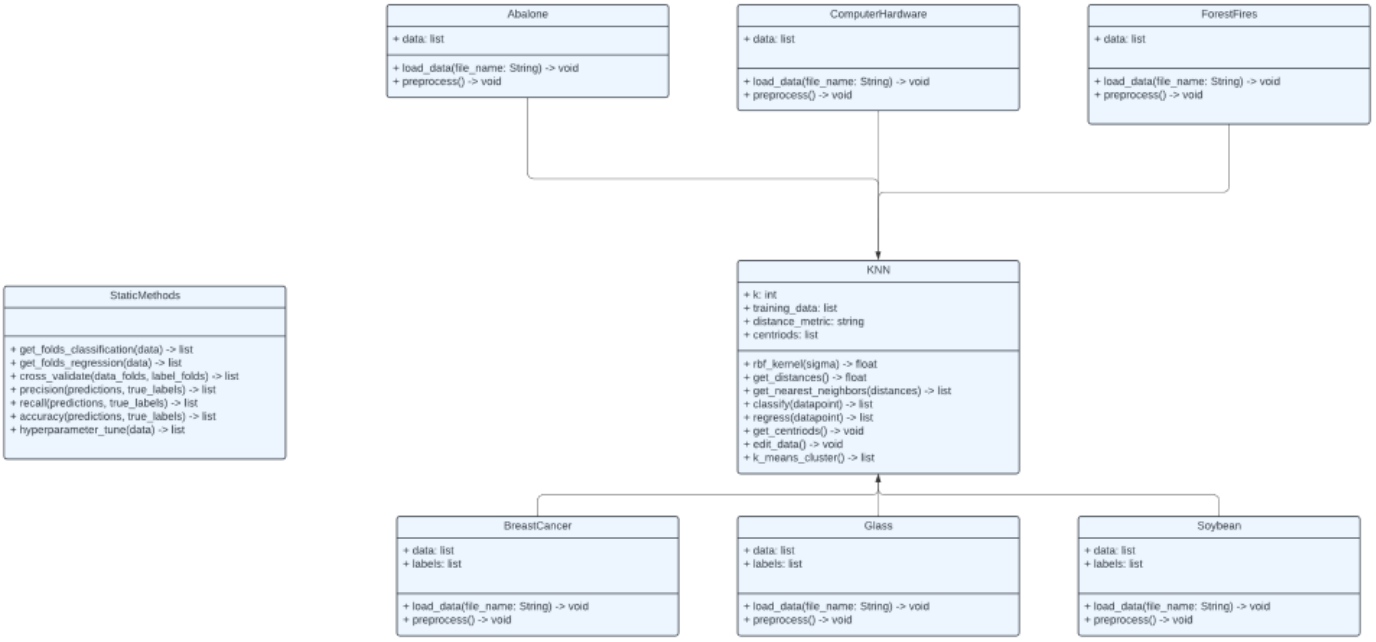

Figure 1. UML Class Diagram for Nearest Neighbor Algorithms

KNN Class

- This class contains the entirety of the algorithms that were specified earlier in the design document. It has methods that contain the code for all of the features. The point of this class is to satisfy all the requirements that regard to the actual algorithms that are being implemented in this project.

  The attributes for this class are:
  1. k: This is an int which represents the number of nearest neighbors to use when making a prediction
  2. train_data: This is a list which contains all the data that has been given to the algorithm.
  3. distance_metric: This is a string that will help determine what distance metric to use.
  4. centroids: This is a list of points that represent the centroids for the K-means clustering algorithm.

  The functions for this class are:
  1. rbf_kernel(): This function takes in a float which represents sigma and then it will compute the value after it is inputted to the function and return it. This value will then be used in the regression task to help classify a data point.
  2. get_distances(): This function uses the training_data and it will return a list of distances for the points.
  3. get_nearest_neighbors(): This function uses the training_data and will return a list of nearest neighbors. This will be calculated by using the Minkowski Metrics where the p value will be tuned for each dataset.
  4. classify(): This function will take in a data point from the classification datasets and then it will predict the data's class by getting a majority vote on nearest neighbor classes by using all of the other functions in the class and it will return it.
  5. regress(): This function will take in a data point from the regression datasets and then it will use the radial basis function kernel and the nearest neighbors to calculate the best estimate for the feature value that is specified to be predicted.
  6. get_centroids(): This function will use the train data to get centroids for the K-means clustering algorithm by adjusting them to the center of each cluster until the clusters no longer change. The k in this algorithm will represent clusters not classes and it will need to be tuned. The base for the k will be $\sqrt{number\ of\ samples}$.

Classification Classes (Soybean, Glass, BreastCancer)

- These classes represent all the dataset classes that will be used for the classification problem that is specified. They all will have slight variations in their functions but the core ideas will be the same. The classes just have different preprocessing steps that will be performed.

  The attributes for these classes are:
  1. data: This is a list of all the data points that are given in the dataset.
  2. Labels: This is a list of all the labels that correspond with the data points that are given in the dataset.

  The functions for these classes are:
  1. load_data(): This function takes in a file path and it will load the data from the csv file into the numpy array.
  2. preprocess(): This function will perform any preprocessing steps that are required for the specific dataset. For example, this could be the handling of missing values or handling of categorical values.

Regression Classes (Abalone, ComputerHardware, ForestFires)

- These classes represent all the dataset classes that will be used for the regression problem that is specified. They all will have slight variations in their functions but the core ideas will be the same. The classes just have different preprocessing steps that will be performed.

  The attributes for these classes are:
  1. data: This is a list of all the data points that are given in the dataset.

  The functions for these classes are:
  1. load_data(): This function will take in a file path and it will then load the data from the csv file into a numpy array.
  2. preprocess(): This function will perform any preprocessing steps that are required for the specific dataset. For example, this could be the handling of missing values or handling of categorical values.

StaticMethods

- This contains all of the helper functions that may be needed to calculate things that are not directly related to the main algorithm or datasets.
  1. recall(): This function takes in the predicted labels and the true labels and returns the recall value.
  2. precision(): This function takes in the predicted labels and the true labels and returns the precision value.
  3. accuracy(): This function takes in the predicted labels and the true labels and returns the raw accuracy value.
  4. get_folds_classification(): This function takes in a set of classification data and returns a list of ten specific folds. The process of stratifying across folds happens in this function.
  5. get_folds_regression(): This function takes in a set of regression data and it will sort the data based on the response value in the data. It will then break the data into groups. Then it will draw the first item from each group for the first dataset and the second for the second dataset.
  6. cross_validate(): This function takes in a set of folds and it will perform 10-fold cross-validation. This function will return a list of metrics to evaluate the model.
  7. hyperparameter_tune(): This function will take in a set of data and tune all the hyperparameters in the model (k, sigma, etc.). It will then return the parameters that performed best on average.

System Flow

To run the system and make sure that all the requirements are fulfilled, start by creating all of the datasets that are shown in the class diagram. Then create a separate KNN class for each dataset. The datasets are grouped into classification datasets and regression datasets so it is recommended that the KNN classes are grouped similarly. Next, for each class, you will run the get_folds() function from the StaticMethods file. This will get ten folds and you will use one of them as the tuning set for the hyperparameters. This fold will then be passed into the hyperparameter_tune() function. After this is done, concatenate the other nine folds and rerun get_folds() again to get ten new folds from the original nine folds. Now, call the cross_validate() function and pass in the ten new folds. This function will use the evaluate() or regress() methods to classify or predict a value for the test dataset. The evaluate() and regress() methods use all of the other functions in the KNN class to give predictions on the test data. Then once all the points in the test dataset have been predicted, the cross_validate() function will return a list of metrics to evaluate the model. Once this is done for every dataset, this will complete all the requirements for the supervised KNN algorithm. Then you will complete this same process but you will use the edit_data() function in the KNN

class to get an edited dataset for the edited-KNN algorithm. This will complete the requirements for the edited-KNN algorithm.

Next you will need to complete the requirements for the K-means clustering algorithm, you will need to pass in each dataset to the get_cetroids() function in the KNN class specific for the dataset. This will calculate centroids for the dataset. To do this at a large scale, use the cross_validate() to test all the folds by using the centroids that are calculated. Finally you will use the centroids that were calculated in the get_centroids() function as the edited dataset for the base KNN algorithm and then you will follow the steps above to evaluate the model for each dataset. This will complete the final requirement of the program regarding K-means clustering.

Test Strategy

To ensure that we meet project requirements, we will first validate that our data has been effectively preprocessed. This means that our datasets contain only **complete instances** and our categorical values have been **encoded to binary values**. This will be quite simple to validate, as our algorithm will only run if this condition is met.

To validate that the **edited K-nearest neighbors** and **K-means clustering** algorithms are implemented correctly, we will check that the implementation matches our planned system flow and architecture, and will check its performance against our stated performance metrics, ensuring that reasonable performance is attained.

To test that our **grid search hyperparameter tuning** is working as intended, we will print values of the hyperparameters as we run the algorithm, checking to see that it changes values, moving through the whole grid. Further, we will ensure that the hyperparameter values that enable the best performance
To ensure that **noise-making** functioned as intended, we will check that the original dataset and the noisy dataset have a reasonable number of distinct values.

Upon obtaining our results, we will check to make sure that **precision, accuracy, recall, and F1 score** have been calculated correctly by printing the intermediate values used to calculate these metrics and carrying out the calculations manually.

We will check that **10-fold cross-validation** works as intended by ensuring that we get different results on different trials of the experiment and that we get reasonable results, and that we run 10 trials of tuning using different folds for each run.

Task Assignments and Schedule

Throughout the course of the project, each team member is expected to complete their work that is assigned in a timely manner. On the parts of the project that require team members to meet in person, it is expected that the team members will communicate to make a time work where they can meet and complete the task at hand. This communication can be done directly with the other team members since there are only three members in the team. All of the code that is written in this project will be uploaded to a team repository so that it can be accessed at any time if the other team members need it. It is expected that team members will keep their code up to date in the repository if they have made any changes.

<u>Task Descriptions</u>

1. Create Design Document - Follow the Creating a Design Document PDF and on D2L to create a design document specific to project 3.

2. Create Hypothesis - Using the descriptions of the datasets and of the k-Nearest Neighbors algorithm, create a hypothesis about how the different datasets will perform according to the chosen metrics.
3. Update Classification Dataset Classes' Methods - Using classes from project 1 as a template, implement any new methods needed for Nearest Neighbors according to the description of its methods in the Project 2 design document. Specifically, implement new preprocessing methods. After this task, these classes should be ready for use.
4. Create Regression Classes - Follow the description of the regression classes in the Project 2 Design Document to create the dataset classes for Abalone, ComputerHardware, and ForestFires datasets. At this stage, the classes only need their structure; methods do not need to be implemented.
5. Implement Methods for Regression Classes - Follow the method descriptions for the NaiveBayes Class in the Project 2 Design Document. After this task, these classes should be ready for use.
6. Create KNN Class - Follow the description of the KNN class in the Project 2 Design Document. At this stage, the class only needs its structure; methods do not need to be implemented.
7. Implement Methods for KNN Class - Follow the method descriptions for the KNN class in the Project 2 Design Document. After this task is complete, the class should be fully utilizable.
8. Update StaticMethods - Using the StaticMethods from project 1, implement any new methods needed for methods needed for Nearest Neighbors according to the description of its methods in the Project 2 design document. Specifically, implement get_folds_regression() and hyperparameter_tune(). After this task, these methods should be ready for use.
9. Final Code Check - For this task, the team members will need to meet in person to make sure all the code runs as expected so that the data that gets collected later in the project is reliable. Further, team members will check that all code is well-commented and readable
10. Collect Results and Create Figures - For this task, one team member will need to create methods for generating figures from the results of our experiment and run the code to generate those figures.
11. Create a Rough Draft of the Paper - Each team member will be responsible for different sections of the paper, using course materials, the hypothesis created in the beginning of the experiment, and the results generated at the end of the experiment to complete the paper's different sections.
12. Complete the Paper - Team members will review one another's work on the paper and ensure that it meets all standards laid out on the project description on D2L, addresses the hypothesis, and accurately interprets the results.
13. Plan Video - This task requires the general layout of the video to be complete and any "dummy methods" to be implemented. This allows the actual recording of the video to go much smoother and prevents last minute changes while trying to record.
14. Record Video - For this task, a team member will record the video. Another team member will edit the video if necessary. This process will also be used as a final check for anything missing in the project. After this task is complete, the paper, video, and code will be submitted

Figure 2. Gantt chart for schedule of tasks on Nearest Neighbors project.