Journal of Machine Learning Research Format (2019) 7 Submitted 10/11; Published 10/11

CSCI 447: Machine Learning – Project 2

Peter Ottsen PETER.K.OTTSEN@GMAIL.COM
Bruce Clark BRUCEWESTONMT@GMAIL.COM
Forest Edwards FORESTJEDWARDS@GMAIL.COM
Justin McGowen MCGOWEN.JUSTIN.P@GMAIL.COM

Editor: None

Abstract

This paper contains the algorithm, experimental approach, results, and discussion surrounding five different algorithms: k-nearest neighbor (Stefan Aeberhard, 1985), edited k-nearest neighbor (Kazuo Hattori, 2000), condensed k-nearest neighbor (K. Gowda, 1997), k-means clustering where the centroids are used in k-nearest neighbor (Paul Bradley, 1998), and k-medoids where the medoids are used in k-nearest neighbor (Hae-Sang Park, 2006). We wanted to run different experimental k values through each of these algorithms to see if we could learn something about which K values would be best in certain circumstances. First we had to clean and process our data, then We ran through each algorithm, with six different data sets, for everyone of our experimental k values, while doing 5-fold validation to generate an F-score and precision and accuracy measurements. We were able to predict classes for our larger sets with pretty good accuracy for the methods we used compared to the smaller sets.

1. Problem Statement

Searching for trends and patterns in data can prove to be quite a difficult and time-consuming challenge for data analysts, especially when dealing with very large sets of data. An automated method of analysis can be far more beneficial for finding data patterns. In order to develop a model, we use five algoirthms: k-nearest neighbor (Stefan Aeberhard, 1985), edited k-nearest neighbor (Kazuo Hattori, 2000), condensed k-nearest neighbor (K. Gowda, 1997), k-means clustering (Paul Bradley, 1998), and k-medoids (Hae-Sang Park, 2006).

We also are going to experiment with different k-vales for each algorithm across our 6 data sets: Abalone (Division, 1995), car (Bohanec, 1997), forest fires (Cortez, 2007), machine (Phillip Ein-Dor), segmentation (Vision Group, 1990), and a wine quality data set (Paulo Cortez, 2009). We are using the wine data set as a combination of winequality-red and winequality-white.

K-value selection can be a trial and error process. For the purposes of this project we were told to select three different values for k to test. A few guidelines for initial guesses were found to be: select a k that is not a multiple of the number of classes and choose $k = (n)^{1/2}$ as a starting point (P. Hall and Samworth, 2008). Based on not wanting a multiple of the number of classes and taking into consideration the size of our data sets ($n_{min} = 200$ and $n_{max} = 4100$), we decided to use prime numbers to be our k values and chose $k = 13, 37, 67$ as our starting point for analysis. We hypothesize that larger k-values will typically work better on larger data set and smaller k-values will typically work better

on smaller data sets. Since we have data sets ranging from 200 examples to 6000, we choose k-values that had a fairly large span.

2. Description of Algorithms implemented

2.1 K-Nearest-Neighbor algorithm

The K-Nearest-Neighbor algorithm is a supervised learning algorithm that takes in a value K, a training data set X, and a test data set Y. The algorithm goes through each test point y_i in Y and calculates Euclidean distance (equation below) to each training data point x_j in X (for the sake of time complexity of our program, we omit the square root in the Euclidean distance formula as it will return the same nearest neighbors). The Euclidean distance formula:

$$Distance = \sqrt{(a_i{}^2 - a_j{}^2) + (b_i{}^2 - b_j{}^2) + (c_i{}^2 - c_j{}^2)}$$

where a, b, and c are the attributes of the data points, and i denote the test data point and j denotes the training data point.

The distances are then ordered by magnitude and the K lowest distances are selected as the nearest points. The most frequently occurring class in the set of K nearest points for categorical values or takes the average of the average of the class values for discrete data sets, as our guessed value for the class of data point, y_i. That guess gets paired with the actual class of y_i. The algorithm repeats the process for every element in Y, and returns a set of pairs containing our guesses and the actual classes.

2.2 Edited-K-Nearest-Neighbor

The Edited-K-Nearest-Neighbor algorithm takes a set value K, and a training data set X. For each data point in X, it finds the K nearest points, then tries to classify this point based on the highest occurring class in the set of nearest points, just like KNN. Edited KNN then compares the guess with the actual class of the item in our training set. If the classes are different, that point gets removed from our training set. The algorithm repeats the process through the entire training set until no other points are removed, and then it returns the reduced training set. This method will remove points from the training data that are difficult to classify so when we test our test data against it. After we have reduced our training set, we will run KNN to classify the test data with our new set.

2.3 Condensed-K-Nearest-Neighbor

The Condensed-K-Nearest-Neighbor algorithm works similarly to Edited KNN but will instead only keep points that are difficult to classify in the training data. It takes one of our values of K, and a training data set X. For each data point in X, the algorithm finds the K nearest points to it and guesses the class by finding the highest class occurrence in the nearest neighbors. When the algorithm compares the guess with the point's actual class, rather than removing these points that it miss-classifies (like in Edited KNN), it adds them to another set called Condensed. The algorithm then keeps iterating this process through the training set until nothing else is added to the condensed set.

2.4 K-Means

The K-Means algorithm develops a condensed data set from the training data. The goal is to find a smaller set that accurately represents your data reduce noise and optimize runtime. The K-Means algorithm runs according to the following rules:(Paul Bradley, 1998)

1. We start by by selecting a random number of points within the range of our dataset to be our "centroids."
2. Then for each point inside our training dataset, we find the nearest centroid and assign that point to that cluster. For every centroid, we will have a "cluster" of points around it.
3. Once we have assigned each point to a centroid, we move each centroid to the mean of the of the cluster associated with it.
4. Now, we will reclassify each point in our data to the centroid it is closest to again. When the centers of the clusters move, different points may be classified to them.
5. Then repeat step 3 and 4 until convergence and the centroids no longer move. When convergence is achieved, we will find the most commonly occurring class value from the 5 closest neighbors to the centroid and classify that centroid as this value. We return the set of our centroids after convergence is achieved. We can now use these centroids as our training set for our KNN algorithm. For each point in our test data, we find which centroid is closest to that point and classify according to that centroid.

2.5 Partitions Around Medoids

Much like the K-Means algorithm, Partitioning Around Medoids algorithm develops a condensed data set from the training data with the goal to find a smaller set that accurately represents your data reduce noise and optimize runtime. The process is as follows:(Hae-Sang Park, 2006)

1. We start by selecting our "medoids" randomly from out training data.
2. For each point inside our training dataset, we find the nearest medoid and assign that point X_i to it. This gives you a cluster of points assigned to each medoid.
3. Once you have assigned each point x_i to a medoid, calculate the cost (distance) of the medoid to each assigned point. Then for each assigned point x_i and calculate the distance from that points to every other point in that cluster. If there is a lower cost point, make that point your new medoid.
4. Repeat steps 2 and 3 until the medoid data set does not change in step 3.

3. Experimental Approach

First off, we needed to talk about how we processed our data. The function **load_data** will take the input data file as a .csv and convert it into a Pandas dataframe. The function

shuffle_data will randomly reorder all the rows of the dataframe. The function **clean_data** will replace the strings in our dataframe with corresponding numbers for distance calculation. We also normalized all of our data points by finding the total range of our column and mapping values between 0 and 1. The function **slice_data** will split the data into a specified number of sections for 10 fold validation and for algorithms in which we will only use 1/4th of the data.

For our evaluation method, we chose to use 5-fold cross validation. We decided to use five folds to optimize the time it takes to run our program, while maintaining an effective evaluation of our algorithms. We used two methods of evaluating the accuracy of our algorithms based on which data set we use. We used an f-score to evaluate the algorithms when they were run on the abalone, car, and segmentation data sets because these are classification sets. We used regression to evaluate the algorithms when they were run on the forestfires, machine, and wine data sets because they are regression sets. The goal of our algorithm for these sets was to closely approximate the discrete value of the 'class', so we used mean-squared-error determine how far from the actual value our algorithm approximated.

Calculating F-scores: For our data sets with categorical classes, we calculate an f-score based on how frequently we made a correct guess on the class. To do this, we created a confusion matrix. To calculate the f-score from our confusion matrix, we do the following:

$$Fscore = 2 * \frac{precision/recall}{precision+recall}$$

$$precision = \frac{TruePositive}{TruePositive+FalsePositive}$$

$$recall = \frac{TruePositive}{TruePositive+FalseNegative}$$

Because confusion matrices are calculated on a per-class basis for multi-class problems, we computed precision, recall, and Fscore for each class and then averaged these over all classes for each given k value.

The F-score gives us a measure of accuracy in our algorithms. It is a decimal-point value ranging from 0(Lowest accuracy) to 1(highest accuracy).

Some of our data sets have non-discrete class values. We used a mean squared error evaluation to measure the accuracy of our algorithms using these data sets. The mean squared error was calculated as such:

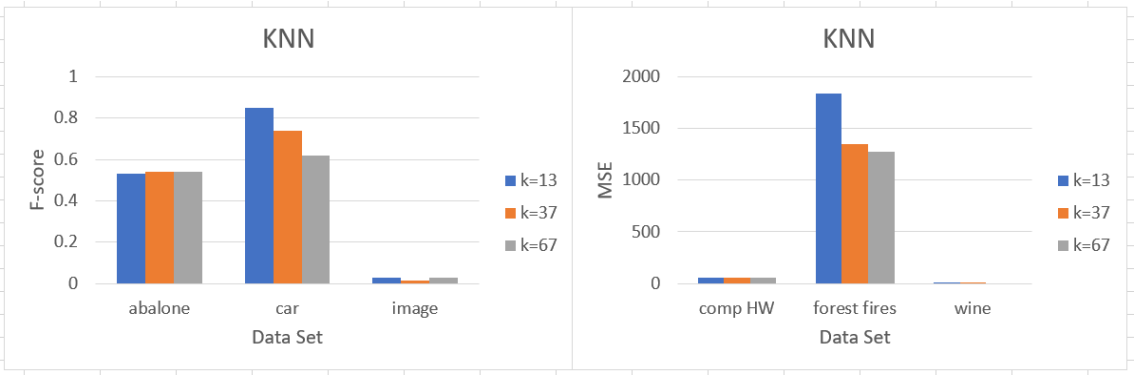
$$MSE = \frac{\sum_{j=1}^{n}(ActualResult_j - PredictedResult_j)^2}{n}$$

Mean-squared-error gives us a positive value that represents the average squared value between the actual and predicted value. Mean-squared-error penalizes more heavily for occasionally guessing values that are further away from the average than consistently guessing values that only vary slightly from the actual value.

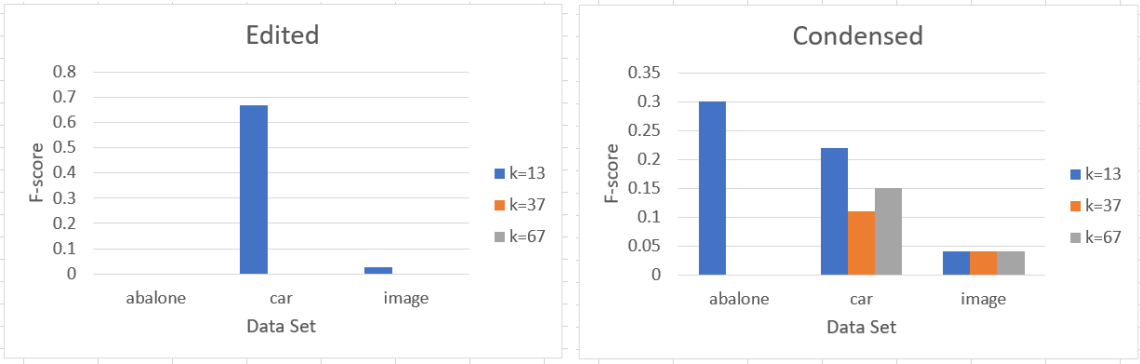
4. Results

For base KNN, we observed that for the large data sets were less affected by the values of K. For the car data set, which was only about 200 entries, we noticed that increasing the

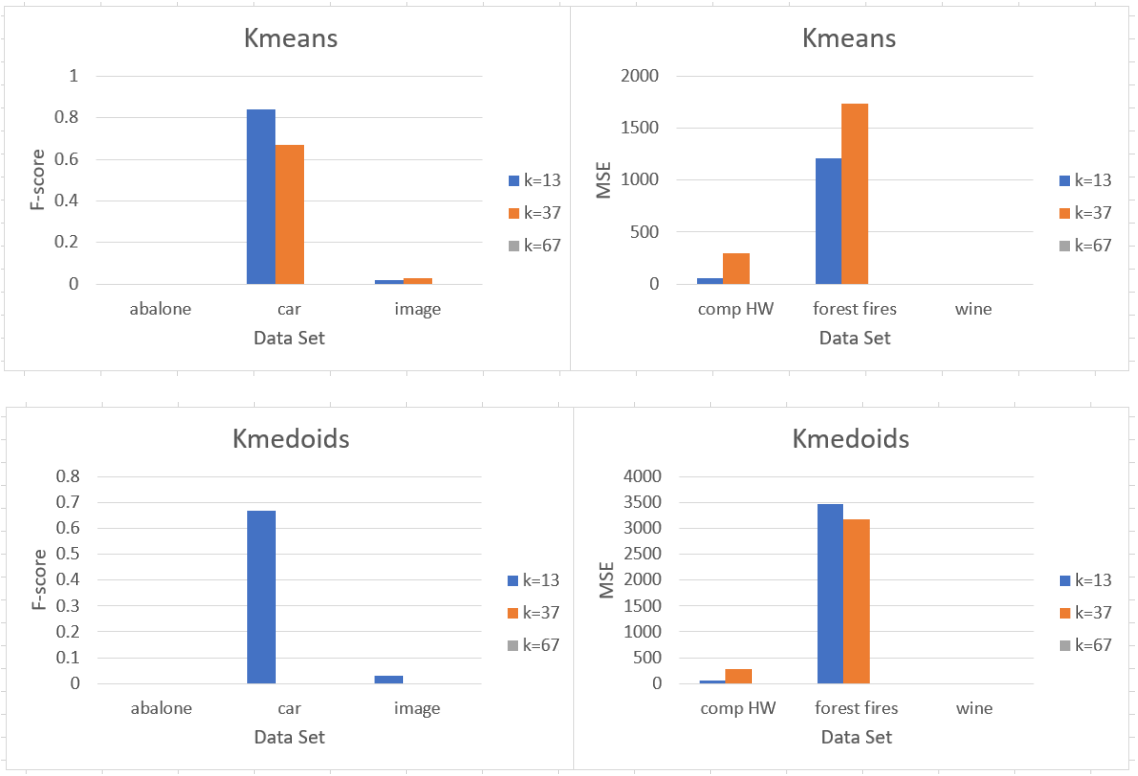
value of K decreased our algorithms efficiency. For forest fires, which has over 500 records, we noticed that an increased k value decreased the mean squared error. For all of the sets where regression was required, increased k would decrease our error. Wine, which was the largest set, had a low error score which is expected as the larger the data set, the more precise we can be with prediction.



We did not allocate enough time to run all of our edited nearest neighbor, so we do not have enough data to understand how this algorithm affects our prediction, although it looks comparable to KNN. Our results for Condensed Nearest Neighbor were very low, regardless of k.



For k means, we see that an increased value of k causes our model to predict less accurately. This is most likely due to the way we classify each cluster. In testing k-means, we noticed that our centroids did not always converge to where the clusters were dominated to one class. To help improve this fact, we could eventually approach labeling the centroids with multiple labels, but that was not something we tested in this algorithm.



5. Discussion

While we were able to get all our algorithms implemented successfully, we ran into a few issues along the way. Our biggest issue had to do with time complexity. Many of our algorithms were not efficient, which proved to be problematic when running them on the larger wine and abalone data sets. It was easy to test the smaller data sets as they would run in just a few minutes, but the larger data sets took hours to complete. For this reason only some of the data sets were completed.

Another issue we were having was that we were getting very low f-scores. This most likely could be solved through further optimization of our algorithms and k-values chosen. Further time to optimize our reduced datasets to ensure that we have the optimal selection of points should lead to better classifications. If more k-values were tested, optimal k-values could be found for each data set.

The other thing that could explain our poor results is our data processing when we change from string values to numbers to calculate distance. For example January and February are right next to each other. So if we assign January to 1 and February to 2 the numbers will properly represent this data. However, when looking at December (12) and January (1), these numbers do not accurately represent the distance between the months. This will be the same when we are looking at days of the week. These factors could contribute to low F-scores.

Another thing our team was surprised by was was just how long it took to run these algorithms. We tried to account for the extended run time of these algorithms, but we underestimated just how long it would actually take. A possible explanation for the extended

run time of our algorithms would be our use of the Pandas library. Even after we performed our data preprocessing we kept our data inside a pandas dataframe. This might have bogged down our algorithms as we were accessing individual numbers in our dataframe consistently, and panda dataframes were not built for being iterated through again and again.

6. Summary

Looking at our results for KNN on abalone: our f-score seems to slightly improve as our K-values get greater. In our KNN car dataset, performance seems to degrade with higher k-values. In the KNN image dataset, we can see that our f-score did not seems to be directly affected by our k-value. Looking at the $\text{Comp}_H W(machine) dataset run on KNN, our MSE(mean square d error) i$
$values got larger. With the wine dataset, we have very low MSE, however, this one also took the longest to run, and w$
$value of 64 with KNN. Looking at our results, we can see that, generally without KNN algorithm, if the dataset wa$
$value and lower MSE. The opposite can also be observed : with smaller datasets, we saw a our best per formance fro$
$vales, in our case k = 13.$

References

Marko Bohanec. Car evaluation database. 1997.

Paulo Cortez. Forest fires. 2007.

Marine Resources Division. Abalone data. 1995.

Chi-Hyuck Jun Hae-Sang Park, Jong-Seok Lee. A k-means-like algorithm for k-medoids clustering and its performance. Technical report, 2006.

G. Krishna K. Gowda. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. Technical report, 1997.

Masahito Takahashi Kazuo Hattori. A new edited k-nearest neighbor rule in the pattern classification problem. Technical report, 2000.

B. Park P. Hall and R. Samworth. Choice of neighbor order in nearest-neighbor classification. *The Annals of Statistics*, 2008.

Usama Fayyad Paul Bradley. Refining initial points for k-means clustering. Technical report, 1998.

Fernando Almeida Telmo Matos Jose Reis Paulo Cortez, Antonio Cerdeira. Wine quality. 2009.

Jacob Feldmesser Phillip Ein-Dor.

James Givens Stefan Aeberhard, Michael Gray. A fuzzy k-nearest neighbor algorithm. Technical report, 1985.

University of Massachusetts Vision Group. Image segmentation data. 1990.