Performance Effects of Diverse Data Sets on K-Nearest Neighbor Algorithms

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

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

Abstract

In this project, we explore multiple k-nearest neighbor learning algorithms to find how diverse data sets affect the performance of this type of algorithm. We picked the K-Nearest Neighbor, Edited K-Nearest Neighbor, and K-Means Clustering algorithms to test our six data sets. Three of these datasets are classification datasets and three are regression datasets. These datasets are varied in their feature counts, feature ranges, number of classes, target value ranges, and normality. So, we discovered how both the type and variation of a dataset affect our three chosen algorithms. To evaluate our algorithm, we used 0/1 Loss and an F_1 score for classification datasets and Mean Squared Error for regression datasets. We implemented a 10-fold cross-validation method to randomize our data and negate patterns present. We found that the breast cancer, soybeans, and abalone datasets performed best because they are well-scaled, balanced, low-noise, low-outlier datasets. We found that the other three datasets performed poorly because they were poorly scaled, skewed, high-noise, and high-outlier datasets. We found that the three algorithms gave very similar results.

Keywords: K-nearest neighbor, clustering, cross validation, regression, and classification

1 Introduction

In this project, we explore the process of choosing a learning algorithm and how different learning algorithms can be essential to finding connections or important features within the data. Using the best model can drastically improve your results. This is typically explored freely and is chosen through reasoning, or at least narrowing down what models will be most successful. For this project we were given three with a multitude of different data sets, we expect that our edited k nearest neighbor would be our most successful model. We thought this because it should be successful at removing outliers, but would not oversmooth data that is "close" together. We think that overall this would be true and depending on class relationships any of the three could be most successful consistently. We also expect the algorithms to work well on small, low-dimension, well-scaled, balanced, low-noise, low-outlier datasets when compared to larger, higher-dimension, poorly scaled, skewed, high-noise, high-outlier datasets. So, since the breast cancer, soybean, and abalone data sets meet most of the criteria for the algorithm to work well, we expect all three

algorithms to perform relatively well on these data sets. The other three data sets are poorly scaled, skewed, high-noise, and high-outlier, so we expect them to perform generally poorly. However, we expect them to perform better on EKNN and K-Means than on KNN as these algorithms help to remove these poor qualities. To evaluate our algorithm, we used 0/1 Loss and an F_1 score for classification datasets and Mean Squared Error for regression datasets.

2 Software Design

Here we will explain our approach to the project. This means we will go into the significance of the dataset structure, our program design, the reasons behind our evaluation metrics, how our learning algorithms work, and a thoughtful explanation of our experimental design.

2.1 Drivers

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, and 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, F1 scores, and the mean squared error for each fold. These functions are calculated and printed out, displaying the average across all folds for these loss functions. It is important to note that in the regression datasets, one hot coding was not used because we did not assume there to be a strong correlation between the categorical features and outputs in these datasets. In terms of the computer database, the vendor names and model names had many points that were overestimated, underestimated, and were accurate within most of each category. Plus, when looking at the fire data, it was difficult to connect if the months and days had a strong effect on the data output when laid out. For both of these scenarios, we felt it would be more effective generally to leave them out of our considerations. It is important to also note that we did not find significance in our categorical data within our forest fire, abalone, and computer performance datasets because there did not seem to be a strong correlation between that feature value and the output. After this finding, we did not include this information within our regression model.

2.2 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 performs the k-nearest neighbor algorithm by using a Euclidean distance helper method and a plurality vote (done in the mostCommonLabel helper method) to 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 isPrediction-Correct 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. 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.

2.3 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.

2.4 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.

2.5 Mean Squared Error

Mean Squared Error is a loss function used in regression tasks. It measures the average of the squares of the differences between the predicted values and the actual values. It is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of data points. Mean squared error penalizes larger errors more heavily than smaller ones due to squaring the error value. This makes the loss function useful in measuring prediction inaccuracies.

2.6 F1 Score

The F1 score is a measure of accuracy used in classification tasks. It balances the trade-off between precision and recall. It is defined as:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where Precision is the ratio of correctly predicted positive values over the total number of positive values, represented by:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall is the ratio of correctly predicted positive values divided by all values in that specific class, which is represented by:

$$\text{Recall} = \frac{TP}{TP + FN}$$

The F1 score is useful in situations where the class distribution is imbalanced, to help identify over or underfitting.

2.7 0/1 Loss

This algorithm takes all of the guesses and records them as a 0 when they are incorrect and a 1 when they are correct. Then once all of the predictions are made the total incorrect guesses are divided by the total number of guesses. This gives us a percentage of incorrect guesses that will quickly tell us how accurate the algorithm is in predicting classes based on the feature information and more precisely the percentage of estimating the class incorrectly.

$$L(G) = \frac{\sum I}{T} * 100$$

where L(G) is our loss percentage (Incorrect Guessing percentage), G is our Guess, T is the total number of guesses, and $\sum I$ is our total number of incorrect guesses.

3 Experimental Approach

Our test strategy consists of running each 10-fold representation five times so that we have 50 total tests to record averages from. 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. We decided to use a similar strategy in finding our σ value, by testing different values of 10 raised to the negative one through three.

4 Results

This is our results section where we will represent the data that was found from our experiment. Reflection on the data will be expressed in the discussion subsection below all of the tables.

4.1 Breast Cancer Identification Results

This database from Wisconsin Breast Cancer (January 8, 1991) shows data related to tumors found in breasts. Each tumor instance has 9 feature values that are integers on a scale from one to ten and a feature that tells us whether the tumor is malignant or benign through a specified integer value. This was one of our larger datasets, as there are about 700 instances of tumors, with about 450 cases being benign and about 250 cases being malignant. Plus, there are a few instances where the data is missing some attribute within a feature.

KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0014	0.9979	0.0486	0.9280
2	0.0246	0.9628	0.0571	0.9118
3	0.0072	0.9890	0.0383	0.9442
5	0.0174	0.9743	0.0325	0.9530
10	0.0275	0.9587	0.0341	0.9499

Edited KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0188	0.9719	0.0354	0.9490
2	0.0435	0.9333	0.0414	0.9375
3	0.0261	0.9608	0.0325	0.9527
5	0.0188	0.9721	0.0327	0.9519
10	0.0391	0.9403	0.0385	0.9427

Table 1: Figure 3.1.1(KNN) and Figure 3.1.2(E-KNN)
(F_1 is calculated for class malignant)

K-Means				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0087	0.9871	0.0455	0.9331
2	0.0522	0.9185	0.0727	0.8845
3	0.0348	0.9473	0.0385	0.9436
5	0.0203	0.9698	0.0285	0.9587
10	0.0333	0.9496	0.0370	0.9448

Table 2: Figure 3.1.3 (K-Means with 600 clusters)
(F_1 is calculated for class malignant)

4.2 Soybean Identification Results

The Small Soybean Database (1987) contains information about diseased soybeans. The dataset has only 47 instances, with 35 features represented by integers. Four possible classes are represented by D4 (17 cases), D3 (10 cases), D2 (10 cases), and D1 (10 cases). It is also important to note that we have no F_1 score for this data because not all of our folds contained all classes and because of this we were not able to calculate precision and recall.

KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0000	1.0000	0.0250	1.0000
2	0.0000	1.0000	0.0250	1.0000
3	0.0000	1.0000	0.0000	1.0000
5	0.0000	1.0000	0.0250	1.0000
10	0.2000	1.0000	0.1500	1.0000

Edited KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0000	1.0000	0.0250	1.0000
2	0.0000	1.0000	0.0250	1.0000
3	0.0000	1.0000	0.0000	1.0000
5	0.0000	1.0000	0.0250	1.0000
10	0.2250	1.0000	0.2250	1.0000

Table 3: Figure 3.2.1(KNN) and Figure 3.2.2(E-KNN)
(F_1 is calculated for class D1)

K-Means				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0000	1.0000	0.0000	1.0000
2	0.0500	1.0000	0.0750	1.0000
3	0.0250	1.0000	0.0250	1.0000
5	0.1250	1.0000	0.0750	1.0000
10	0.3500	NaN	0.2341	NaN

Table 4: Figure 3.2.3 (K-Means with 41 clusters)
(F_1 is calculated for class D1)

4.3 Glass Identification Results

The Glass Identification Database (1987), used in forensic science, contained information about elements present in glass and its refractive index. Based on the feature data, the class specified what type of glass was present. This database had no missing attribute values, with 214 total instances.

KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0278	0.9714	0.2904	0.7186
2	0.1278	0.8682	0.3165	0.7009
3	0.1778	0.7962	0.3167	0.6870
5	0.2278	0.7941	0.3205	0.7107
10	0.3722	0.6429	0.3316	0.7127

Edited KNN				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.1500	0.8262	0.3092	0.7073
2	0.2889	0.7540	0.3241	0.7169
3	0.3167	0.7000	0.3558	0.6785
5	0.4111	0.6286	0.3447	0.6795
10	0.4778	0.5300	0.3688	0.6688

Table 5: Figure 3.3.1(KNN) and Figure 3.3.2(E-KNN)
(F_1 is calculated for class building windows float processed)

K-Means				
K	$0/1_{tuning}$	$F_{1tuning}$	$0/1_{testing}$	$F_{1testing}$
1	0.0667	0.9542	0.3203	0.6636
2	0.2500	0.8265	0.3483	0.6940
3	0.3056	0.6804	0.3558	0.6993
5	0.4000	0.6489	0.3335	0.7109
10	0.3944	0.6625	0.3391	0.6942

Table 6: Figure 3.3.3 with 133 clusters
(F_1 is calculated for class building windows float processed)

4.4 Computer Hardware Results

The computer hardware data set described the performance of general computers based on performance metrics of parts. Within this dataset, there are 209 pieces of data with no missing features. There are a total of 10 attributes including vendor name and model.

KNN			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	1	1322.1	16138.7
1	10	550.0	11451.0
1	1000	550.5	3922.6
2	1	1146.8	16088.5
2	10	415.6	11210.9
2	1000	622.3	4354.6
3	1	1047.8	16072.8
3	10	318.5	11167.0
3	1000	716.4	4446.0
5	1	1047.8	16072.9
5	10	335.6	11149.8
5	1000	677.1	4743.6
10	1	1047.8	16072.9
10	10	337.6	11148.2
10	1000	1174.8	4897.5

Edited KNN				
K	σ	ϵ	MSE_{tuning}	$MSE_{testing}$
3	1000	20	2057.9	14862.9
3	1000	40	1082.5	10264.3
3	1000	60	927.4	10188.5
3	1000	80	672.8	9219.0
3	1000	100	735.9	8984.9

Table 7: Figure 3.1.1(KNN) and Figure 3.1.2(E-KNN)

K-Means			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	1000	372.6	3245.2
2	1000	554.4	5228.1
3	1000	714.2	4310.4
5	1000	1104.7	3606.2
10	1000	1096.3	4041.5

Table 8: Figure 3.1.3 with 175 clusters

4.5 Abalone Results

The abalone dataset describes information that is useful in attempting to determine the age of the animal. 4177 pieces of data have no missing feature values. There are a total of 9 features that describe the animal including sex.

KNN			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	1	1.4707	8.3747
1	10	1.4707	8.3747
1	1000	1.4707	8.3747
2	1	3.0390	6.2706
2	10	3.0412	6.2706
2	1000	3.0412	6.2706
3	1	3.5306	5.6707
3	10	3.5335	5.6707
3	1000	3.5335	5.6707
5	1	4.2131	5.0893
5	10	4.2164	5.0894
5	1000	4.2164	5.0894
10	1	4.5569	4.7497
10	10	4.5598	4.7499
10	1000	4.5598	4.7499

Editied KNN				
K	σ	ϵ	MSE_{tuning}	$MSE_{testing}$
10	10	1	5.3697	5.1755
10	10	2	5.2112	5.0515
10	10	3	5.2975	5.0841
10	10	4	5.1599	4.9575
10	10	5	4.9133	4.8571

Table 9: Figure 3.1.1(KNN) and Figure 3.1.2(E-KNN)

K-Means			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	10	2.5099	7.1680
2	10	3.7205	5.8360
3	10	4.0363	5.3203
5	10	4.3354	5.0018
10	10	4.6111	4.9772

Table 10: Figure 3.1.3 with 3500 clusters

4.6 Forest Fire Results

The forest fire data set describes data around a fire, where we are trying to predict how big the fire was. This dataset contained 13 features with no missing information and had 517 pieces of data. It is important to note that this dataset was very skewed as there were strong outliers with large fires.

KNN			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	1	330.7	7271.6
1	10	330.7	7271.2
1	1000	330.7	7271.2
2	1	332.8	7324.0
2	10	617.6	6668.1
2	1000	898.9	6527.8
3	1	349.4	7315.0
3	10	757.7	5455.5
3	1000	893.3	5090.3
5	1	386.6	7314.1
5	10	924.8	4819.0
5	1000	975.5	4480.0
10	1	391.0	7314.1
10	10	1168.7	4604.3
10	1000	1524.2	4321.6

Edited KNN				
K	σ	ϵ	MSE_{tuning}	$MSE_{testing}$
10	1000	5	938.2	4135.3
10	1000	10	952.9	4117.0
10	1000	15	950.4	4107.3
10	1000	20	950.6	4096.2
10	1000	25	975.6	4102.2

Table 11: Figure 3.1.1(KNN) and Figure 3.1.2(E-KNN)

K-Means			
K	σ	MSE_{tuning}	$MSE_{testing}$
1	1000	846.9	5530.5
2	1000	1528.6	4906.5
3	1000	1583.8	4970.8
5	1000	1694.9	4296.8
10	1000	1256.8	4077.7

Table 12: Figure 3.1.3 with 388 clusters

5 Summary

Overall, our algorithm was quite successful in classifying but struggled within our regression models. It was interesting to see that our edited k-nearest neighbor was quite similar to our other algorithms and that the difference between them was not as drastic as we expected. We think this is because some data marginally fits a model of slight difference better. This could be explained by the most common relationships between information. For example, Abalone is performing well because the data is scaled between zero and one. It is like the data is almost already normalized for us, again contributing to why our other algorithms may not have been successful with forest fires and computer hardware. As we expected, we found that the breast cancer, soybeans, and abalone data sets did perform best. And, as we expected, we found that the other three data sets performed poorly. However, the difference between the KNN values and the EKNN and K-Means values for these data sets was not as extreme as we thought it would be. It appears that they can only do so much to smooth the data; a bad data set is a bad data set. One clear limitation is always the amount of information present. With more information, more accurate connections between data can be made. Plus, we do not know the exact accuracy of the information as it could be somewhat inaccurate. It is also important to consider the inaccuracy of our hyperparameters to make useful contributions to understanding and predicting our data. The fact that we chose not to include categorical features in the regression data sets may have been a limitation as well; we may have been incorrect in our assumption and they may have provided useful information.

6 Appendix

Max Hymer:
- Paper (abstract, experimental approach, results)
- Code (driver setup, 10-fold cross validation, debugging all three knn algorithms)
- 50% effort

Michael Downs:
- Paper (introduction, software design, summary)
- Code (knn implementation, eknn implementation, k-means implementation, distance/kernel functions, loss functions)
- 50% effort

Combined Work:

- Hyperparameter tuning
- Video commentary and recording
- Did all work together, there was collaboration between most parts