Investigating the Effect of Dataset Reduction in K-Nearest Neighbor Classification and Regression

Carlos Rivas CARIVAS007@GMAIL.COM
Department of Computer Engineering
Montana State University
Bozeman, MT 59715, USA

Finnegan Davidson DAVIDSONFINN@GMAIL.COM
Department of Computer Engineering
Montana State University
Bozeman, MT 59715, USA

Editor: Carlos Rivas and Finnegan Davidson

Abstract

In this study, we investigate the difference in performance of a k-nearest neighbor (kNN) classification/regression learning algorithm under three different circumstances. The first circumstance is without any dataset reduction, the second circumstance is using the edited nearest neighbor (ENN) dataset reduction strategy, and the third is using the k-means clustering (kM-kNN) dataset reduction strategy. To complete this investigation, six experiments were set up to measure the average performance of the kNN algorithm when using each of the three dataset reduction strategies. Each of the six experiments utilize a different dataset, where three of the datasets test classification while the other 3 test regression. Performance is evaluated using multiple metrics to provide a deeper insight into the algorithm's capability. In each experiment, all three strategies of dataset reduction are used and 10-fold cross validation is completed within each strategy. This allows the algorithm to be tested more times on the same dataset to average the results and determine whether there is a relationship between performance and which dataset reduction strategy is used.

1 Introduction

When applying a machine learning algorithm to a problem, it is important to choose the correct model for the problem. Choosing the correct model can have a dramatic effect on algorithm performance and can optimize the computational resources required. To make the optimal choice, it is important to understand how different algorithms perform for a generalized task. For this reason, there is considerable effort being expended within the machine learning community to understand how different algorithms perform. This project contributes to that understanding by looking at the relative performance of three different nearest neighbor classification and regression strategies. We want to determine how well the k-nearest neighbor, edited nearest neighbor, and k-means clustering with k-nearest neighbor strategies handle variations in their training data. To accomplish this, we tested each of these strategies on three classification datasets and three regression datasets. We then evaluated the performance of each algorithm using 0/1 loss and F_1 score for classification,

and mean squared error (MSE) and mean absolute error (MAE) for regression. We predicted that using one of the dataset reducing strategies will decrease kNN prediction performance. The reasoning for this prediction is that the more data kNN has, the more likely a 'near-neighbor' will be present when predicting a data point, therefore leading to a more accurate analysis of the data point. Our experiments provided interesting results, indicating that kNN without dataset reduction had the lowest performance variance, and performed at a high level more consistently than when using a dataset reduction strategy. Described below is the experimental approach, the results of our experiments, and the conclusions we drew based on those results.

2 Experimental Approach and Program Design

Proposed Problem Solution To investigate how reducing the model size affects the performance of a kNN classifier, there are five main areas that need to be considered: data processing, hyper-parameter tuning, dataset reduction, kNN classification or regression, and kNN performance. Within the dataset class, the raw data will be processed to be used for dataset reduction or kNN classification/regression. This will include continuization, normalization, and imputation when necessary as well as folding and splitting. The result of this processing is a 'tune set', which holds 10% of the examples in the dataset to be used for hyper-parameter tuning, and a 'validate set', that contains the other 90% of the examples in the raw data. During this process both the tune set and validate set are stratified to match the class or response value distribution of the raw dataset.

Once the data is processed correctly, it can be used for tuning our hyper-parameters. The process of tuning involves testing the results of the kNN algorithm, as well as both methods of dataset reduction, for a variety of hyper-parameter values. This process helps determine the ideal hyper-parameter values for each dataset, which can then be used by kNN, ENN, and kM-kNN to determine our final results.

With the ideal hyper-parameters chosen, the dataset is ready to be reduced. For our experiments, we used two different methods of dataset reduction. The first method, edited nearest neighbors, removes data points that are incorrectly classified or are outside an error threshold for regression. This effectively reduces noise in the dataset because the examples that aren't correctly classified are no longer considered in future classification/regression tasks. The second method, k-means clustering with k-nearest neighbors, reduces the dataset to a set of centroids. These centroids represent the average location of examples with similar features within the original dataset. During this process the folds determined in data processing are maintained to ensure cross-validation can still be performed.

Once dataset reduction has been performed, the reduce dataset can be used for classification/regression tasks by kNN. To investigate the effect of reducing the dataset on the k-nearest neighbor algorithm's performance, we will run the algorithm on the original validation set, the dataset reduced by ENN, and the dataset reduced using k-means clustering. To ensure our results are valid, we use 10-fold cross validation. To do this, we select one of the ten folds in the dataset, and use the other 9 to create the model. We then use the tenth fold as a hold-out fold and classify or compute the response value for each example in the fold. This process is repeated using each of the 10 folds as the hold-out fold, and the results are averaged to get a more generalized result for the algorithm. We can then

use loss metrics to compare the performance on each dataset and determine which strategy produces the best kNN results.

To evaluate our performance, we selected two classification based loss metrics and two regression based loss metrics. For classification, we used 0/1 loss and F_1 score. For regression, we used mean squared error and mean absolute error. By using these metrics and comparing the results of each strategy, we can determine if the kNN algorithm performs better with a reduced dataset, and which method of dataset reduction produces better results.

Dataset The goal of the dataset class is to prepare the raw data to be used by the algorithms. Since there are 6 different datasets being used for this experiment, the required processing for each set varies slightly. To accomplish all of the required processing for each dataset, we included 11 different methods in the class. These methods provided functionality to continuize data, impute data, shuffle a raw dataset or one of the subsets, sort data by a class or response value, split data into desired subsets, normalize data using min/max normalization, remove an attribute from the data, fold data into the desired partitions, and save or extract data to/from a CSV file. For some of the datasets, there is a non-attribute feature in each example, such as an ID or case number. The method to remove an attribute was used to remove these features. To continuize the data, we chose to take discrete valued attributes or labels, and assign them an integer value. An example of this was in the Abalone dataset, where there is an attribute that can be M, F, or I. To continuize this, we would assign all examples with the attribute M a new value of 0, all examples with the attribute F a new value of 1, and all examples with the attribute I a new value of 2. This process generalizes for other datasets with more possible attribute or label values. To impute the data, we chose to replace the missing feature with a random integer value between 1 and 10. Not every dataset has to utilize each of these methods to be processed correctly, and we only used the necessary methods to prepare each dataset.

- **Classification Sets** The three classification sets required slightly different processing flows to properly prepare the data for classification. For the cancer and glass datasets, we had to remove the ID number found in the first index of each example. For the soybean data, we had to continuize the data to ensure it was in the proper format. Once this had been completed, each dataset was shuffled and sorted based on the class labels. The datasets were then split into the desired subsets. Finally, the validation set was folded and stratified and the tune set was shuffled.

- **Regression Sets** All three of the regression datasets (abalone, forest fire, machine) required the same processing flow in order to be prepared for regression tasks, with the added step of normalizing the value to be regressed. To process these datasets, we continuized the data to ensure it was in the proper format. We then normalized, shuffled and sorted the data based on the response value of each example. After these steps, the data is ready to be split into the desired subsets. Once in the desired subsets, the validation set is folded and stratified and the tune set is shuffled.

Hyper-parameter Tuning Once the data was processed, we needed to tune our hyper-parameters. When analyzing the performance of the kNN algorithm without any dataset

reduction, there are only two hyper-parameters to consider: k_n, which is the number of neighbors to consider, and σ (for regression), which is the bandwidth for the RBF kernel. In our implementation, using the ENN strategy to reduce the dataset creates no additional hyper-parameters, but we could have included the ϵ hyper-parameter, which sets an error bound for what is considered "incorrect regression". Instead of including this, we chose to use a 5% error bound. When using the kM-kNN strategy to reduce the dataset, there is one additional hyper-parameter to consider, which is k_c, the number of clusters to use or centroids to place.

To tune these hyper-parameters, we set up a method to test a variety of different hyper-parameters values and return the value of each hyper-parameter that resulted in the best performance. We test these values over a number of tuning epochs, changing one hyper-parameter value per epoch. The amount we change our hyper-parameter is determined by the increment value, which can be set for each different hyper-parameter. We can also set the number of epochs so that we can reach the largest value that we would like to test, based on the increment. This process is completed for each necessary hyper-parameter.

Dataset Reduction

- **Edited Nearest Neighbors** The edited nearest neighbor dataset reduction strategy reduces a model by removing examples that are misclassified. The process for removing these examples begins by tuning and running kNN. Once we have tuned and ran kNN, we are able to determine which points were incorrectly classified. For classification tasks this means predicting the wrong class for the example and for regression tasks this means not being within 5% of the correct value. Once the examples that weren't placed correctly have been determined, they are removed from the model. This reduced dataset can then be used as the model for the final iteration of kNN. This process is repeated for each model (entire validation set except a hold-out fold), creating 10 models that have been reduced by the ENN process. The results of the final kNN iteration on each of these 10 models can be used to determine if this strategy had an effect on the performance of the algorithm.

- **K-Means Clustering** The k-means clustering dataset reduction strategy reduces a model to a collection of centroids. These centroids represent the average location of each class for classification, or the average location of each response value range for regression. To determine the location of these centroids, we initialize k_c centroids randomly. We then assign each example in a model (entire validation set except a hold-out fold) to the centroid it is located closest to based on the euclidean distance of each of its attributes compared to each centroids attributes. Once each example is assigned to a centroid, the average position of each cluster is calculated. These values become the new centroids, and the process is repeated until the centroids stop moving more than 5% per iteration. Once the centroids have converged, these values are stored and become the reduced dataset to be used as the model for classification or regression. Once one model's centroids are calculated, we repeat the process for all 10 models. Dataset reduction concludes once the centroids have been calculated for each model. The reduced dataset can then be used for a final iteration of kNN to determine if this strategy had an effect on the performance of the algorithm.

k-Nearest Neighbor Classification/Regression For our experiments, we are using a k-nearest neighbor classification and regression algorithm. The process for the algorithm is simple. It begins by storing the validation data as the model. In our implementation, we used nine of the ten folds in the validation set as the model, leaving one hold-out fold to classify or regress. To predict the class or response value of an example, the algorithm calculates the distance of the example compared to each point in the model, using euclidean distance. It then uses the k_n closest examples to predict the class or response value. We used a majority-vote classification system to classify examples and an RBF kernel to predict the response value for regression sets. Once the predictions have been made, the results of the algorithm can be stored and performance can be analyzed.

Performance Evaluation Metrics To evaluate the performance of the kNN algorithm we chose two classification based loss metrics and two regression based loss metrics. For classification we chose 0/1 loss and F_1 score as our metrics. We chose these metrics because they give a generalized performance metric for the algorithm and are simple to compute. For regression we chose mean squared error (MSE) and mean absolute error (MAE) as our performance metrics. We chose these metrics because MSE places a higher emphasis on outliers in the regression performance, whereas MAE places a lower emphasis on outliers which can provide a more accurate depiction of performance depending on data context. By using both these metrics we can evaluate performance with and without an emphasis on outlier data. Calculating loss not only allows us to tune hyper-parameters, but it also allows us to compare the performance of kNN on an unaltered dataset versus the reduced datasets from ENN and kM-kNN. Since all regression values are normalized during pre-processing, a comparison of performance can be made across datasets.

3 Results

3.1 Classification Results

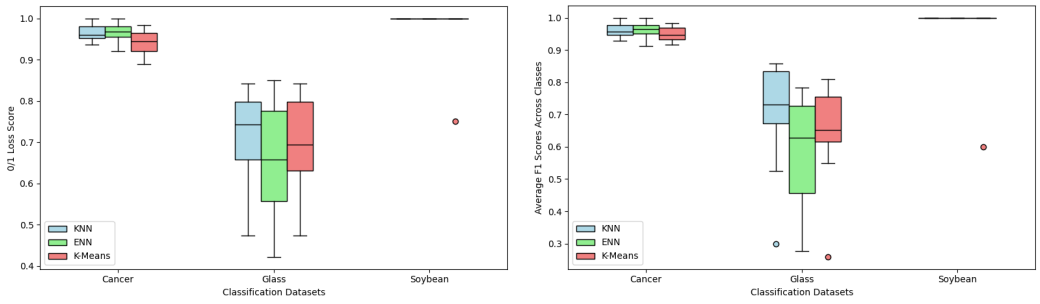

Figure 1: Comparison of 0/1 Loss and F1 Scores for Classification Datasets

Breast Cancer Dataset Classification Results The Wisconsin Breast Cancer Database (January 8, 1991) contains 699 examples of breast cancer diagnoses. Each example has 10 discrete valued features. For training, the ID number in the first index was removed. There are 458 examples of diagnoses involving a benign tumor and 241 examples of diagnoses

involving malignant tumors. There are 16 examples in the dataset with a missing feature value.

	Cancer Database		
	KNN	ENN	K-Means
Average Testing 0/1 Loss	0.96500	0.96502	0.93960
Average Testing F_1	0.96120	0.96144	0.94998

Table 1: Average performance metrics for Cancer dataset.

Glass Dataset Classification Results The Glass Identification Database (September, 1987) contains 214 examples of different types of glass. The examples are distributed into 7 different classes. Each example has 10 features and a class label. Of the 10 features, 9 were used within training with the ID number found in the first index of each example excluded from the data. The 9 training features are all continuously valued and fall within a range specific to each feature. There are no missing features in the dataset.

	Glass Database		
	KNN	ENN	K-Means
Average Testing 0/1 Loss	0.71815	0.65578	0.69763
Average Testing F_1	0.70207	0.57840	0.64404

Table 2: Average performance metrics for Glass dataset.

Soybean Dataset Classification Results The Small Soybean Database (1987) contains 47 examples of diseased soybeans. The examples are distributed across 4 classes with 10 examples each in D1, D2, and D3, and 17 examples in D4. Each example has 35 discrete valued features. There are no missing feature values within the dataset

	Soybean Dataset		
	KNN	ENN	K-Means
Average Testing 0/1 Loss	1.0	1.0	0.975
Average Testing F_1	1.0	1.0	0.96

Table 3: Average performance metrics for Soybean dataset.

3.2 Regression Results

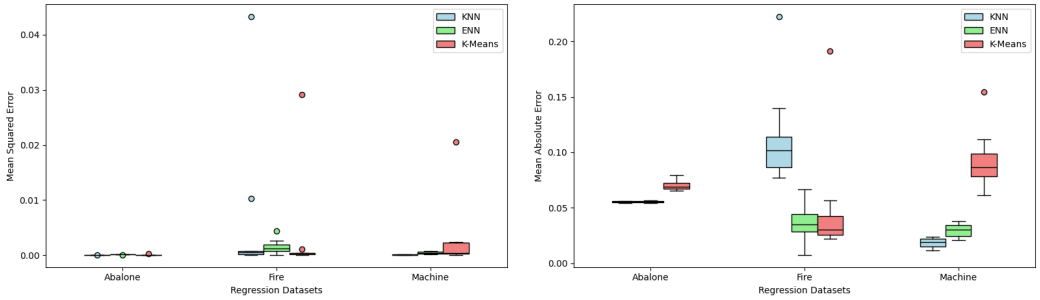

Figure 2: Comparison of MSE and MAE for Regression Datasets

Abalone Dataset Regression Results The Abalone Database (1995) contains 4177 examples of abalone shellfish with the number of rings on their shell as a response value. The examples are distributed across 29 classes with the majority of the examples falling between 5 and 15 rings. Each example has 8 features with 1 nominal and 7 continuous values. There are no missing feature values.

	Abalone Dataset		
	KNN	ENN	K-Means
Average Mean Squared Error	0.00001	0.00015	0.00004
Average Mean Absolute Error	0.05543	0.05536	0.07041

Table 4: Average performance metrics for Abalone dataset.

Forest Fire Dataset Regression Results The Forest Fires Database (2007) contains 517 examples of forest fires with the area burned as a response value. The response values of these examples range from 0.00 to 1090.84 hectares with the examples heavily skewed towards 0.00 hectares. Each example has 12 features with 2 being nominal and the rest continuous. There are no missing feature values in the database.

	Forest Fire Database		
	KNN	ENN	K-Means
Average Mean Squared Error	0.00566	0.00150	0.00322
Average Mean Absolute Error	0.11285	0.03487	0.04893

Table 5: Average performance metrics for Forest Fire dataset.

Machine Dataset Regression Results The Relative CPU Performance Database (1987) contains 209 instances of machines with their estimated relative performance as the response value. The response values range from 15 to 1238 with an average value of 99.3. Each ex-

ample has 9 attributes with 2 nominal features and the rest being continuous. There are no missing feature values.

	Machine Dataset		
	KNN	ENN	K-Means
Average Mean Squared Error	0.00007	0.00043	0.00290
Average Mean Absolute Error	0.01835	0.02967	0.09208

Table 6: Average performance metrics for Machine dataset.

4 Discussion and Conclusions

4.1 Analysis of Results

Classification Starting with classification performance on the cancer dataset, kNN and ENN had virtually the same average performance, with kM-kNN falling behind. It is also worth noting that kNN had the lowest variance of all three algorithms for the cancer dataset. When examining glass classification performance, kNN edges out the other algorithms' average performance scores, with kM-kNN coming in second and ENN coming in last. kM-kNN managed to produce the lowest variance among the three algorithms for the glass dataset. Finally, for the Soybean dataset kNN and ENN both had perfect average performance metrics, where kM-kNN came close except for a single outlier.

The takeaways from these results are that kNN produces the best overall performance, where it's average performance metrics either led or tied the other algorithms for each dataset. It is worth noting the kM-kNN algorithm outperformed kNN in performance variance for the glass dataset F1 score. Based on our results, there is essentially no reason to implement ENN over kNN for classification tasks, especially when considering the added computational complexity of the algorithm. Considering this, a case for kM-kNN implementation could be made for specific situations, as the computational complexity is lower compared to the other two algorithms (due to the severe reduction in dataset size). Additionally, our data shows that in specific situations kM-kNN performance variance can outperform kNN.

Regression Moving to regression performance, kNN significantly outperformed ENN and kM-kNN in terms of average MSE and MAE on the Abalone dataset, although all of the algorithms performed quite well. The fire dataset is the only instance where kNN under performed the other two algorithms, where kNN had significantly higher average MAE performance, and ENN had the lowest average performance metrics overall. The machine dataset trended positively towards kNN, where ENN came in second for performance and kM-kNN came in third.

When looking at the regression data, the takeaways are little more unclear as to which algorithm performed the best overall. The most definitive takeaway is that when considering outlier data (fire dataset), reducing the number of data points can increase performance, where both eNN and kM-kNN outperformed kNN. Considering this, kNN still managed to produce the best results when significant outliers were not present. When comparing the

Abalone and Machine datasets, it could be worth running further experiments to determine if dataset size correlates to reduced performance variance, as the large Abalone dataset had much lower variance when compared to the smaller Machine dataset.

4.2 Limitations

While our experiments were able to provide some insight into the relative performance of kNN using different dataset reduction strategies, there are many ways that our investigation could be improved. One way that we could have improved our results is including more hyper-parameters. We could have added two hyper-parameters within the ENN strategy for data reduction that could have made the algorithm's performance better. The first hyper-parameter that could be added is i_e, the number of editing iterations that the ENN algorithm completes. We chose to use one iteration, so only the points misclassified during the initial kNN classification that ENN completes are removed. Repeating this process might have been optimal. Another hyper-parameter that we could have used to improve our performance is the ϵ hyper-parameter. This hyper-parameter sets the error bound for what is considered 'incorrect regression'. We chose to use a set 5% error bound, but tuning this could result in better performance of the algorithm.

Another limitation of our algorithm was the small range of hyper-parameters tested during tuning. With the strategy we used, tuning is completed by starting at a set value and increasing it by a set increment, a set number of times. This checks a relatively small range of hyper-parameters. This strategy also only tunes one hyper-parameter at a time, so it may not return the optimal combination of hyper-parameter values. We used this strategy to balance the time required for tuning while also still being able to find relatively optimal hyper-parameters.

An additional limitation in our tuning is how we select the best hyper-parameter from the tuning results. Since we have two evaluation metrics, we chose to select the best hyper-parameter by averaging the best performing value for each metric. For example, if during classification for a tuning set, the best 0/1 performance happened with $k_n = 1$ and the best F_1 score happened when $k_n = 7$, the algorithm would select $k_n = 4$ as the optimal hyper-parameter value. This strategy might not find the best hyper-parameter, since it is possible that the algorithm would perform better with $k_n = 1$ or $k_n = 7$ than it would with $k_n = 4$. A different selection strategy could create a more robust testing environment for the investigation.

4.3 Summary

The goal of this project was to implement a kNN classification/regression learning algorithm and evaluate its performance using three different strategies of dataset reduction. Investigating how different dataset reduction strategies affect the performance of nearest neighbor algorithms is important for applications in a wide variety of fields since there are many classification and regression tasks that can be completed using this type of algorithm. To contribute to this investigation and try to deepen the collective understanding of nearest neighbor algorithms in computer science, we developed and tested a k-nearest neighbor multinomial classification and regression algorithm and tested it on six different datasets. To analyze the effect of data reduction, we implemented ENN and kM-kNN to reduce each

dataset before classification or regression. To increase the validity of our results we used 10 fold cross validation and multiple performance metrics to provide a greater number of models to analyze and multiple metrics to compare them. We accomplished this goal using four python classes with one class handling processing, and the other three handling one strategy of dataset reduction each.

Considering the results from the 6 experiments conducted, it can be concluded that kNN had the best overall performance. However, there are specific cases where implementation of the other learning algorithms is advantageous. In general, kNN produced the best average performance metrics and performance variance. The cases for algorithm implementation aside from kNN rely on lower computational complexity for kM-kNN, and reduced error in ENN and kM-kNN due to outliers for regression tasks. Our data supports our original hypothesis, where kNN would perform the best among the 3 learning algorithms.

5 References

Materials and ideas discussed were pulled exclusively from CSCI-447 lecture content and class material.

6 Appendix

Throughout the duration of this project, both group members worked together to complete the necessary tasks, meeting daily to discuss progress and redistribute work when necessary. Both group members feel the work load was equitable and balanced, with neither member providing significantly more than the other to any task.

Task	Completed By:
Design Document	Finn/Carlos
Dataset Class	Finn/Carlos
kNN Class	Carlos
ENN Class	Carlos
kM-kNN Class	Finn
Classification	Finn
Process Verification	Carlos
Project Report	Finn/Carlos
Project Code Video	Finn/Carlos
Project Video Recording	Carlos

Task Delegation