Implementation of Known Nearest Neighbors & Condensed Nearest Neighbors with Cross Fold Validation

Anthony (Tony) Kelly
1-443-717-1215 | akelly31@jhu.edu (Hopkins) kellyt419@gmail.com (primary/personal)
Dr. John Sheppard
EN605.649.81.SP19 Introduction to Machine Learning
Project 3 – KNN, CNN, and CFV

I. Abstract

The following paper highlights the results of the integration effort performed by Tony Kelly in implementing the Known Nearest Neighbors (KNN) algorithm, with a Condensed Nearest Neighbor technique, where both utilized Cross Fold Validation. KNN was employed to attempt to predict classifications for two datasets, the Ecoli and Image Segmentation datasets, and to predict values using regression techniques for two datasets, the Forest Fires and Machines datasets, all from the University of California, Irvine [1].

II. Hypothesis

Based on the results of this paper [2], I would predict that the classification runs of KNN would maintain the same accuracy using CNN but do so with less training points. I would expect that the average number of neighbors to be somewhere near 5 for all classification instances.

I predict that the regression datasets will contain a very high mean squared error, as the attempt to average the values of the k-nearest-neighbors (and described more in section IV) will almost never result in the exact value of the current testing point. Additionally, I anticipate a smaller number of neighbors being needed, again somewhere between 5-10.

III. Description of the Algorithms

A. Known Nearest Neighbors:

Known Nearest Neighbors (KNN) is a relatively simplistic yet effective algorithm used for classification and regression, enabling it to be used in both supervised and unsupervised environments [as performed in this experiment] [3]. It is a non-parametric method, meaning it makes no underlying assumptions about the data. The KNN algorithm, essentially, locates the k-nearest neighbors for a specified testing point using one of many distance functions (Euclidean, Manhattan, etc.). Once those k-neighbors have been identified, the algorithm then either: determines the most frequently occurring class for that newly defined neighborhood, or takes an average of the values (or the simple value we are predicting) for those points (neighbors) in the newly defined neighborhood, for classification and regression, respectively. This technique is instance-based, and makes use of locally defined neighborhoods calculated at run time for each testing point.

B. Condensed Nearest Neighbors:

Condensed Nearest Neighbors CNN is a reduction technique that attempts to reduce the number of training instances needed to maintain KNN performance, thus resulting in less resource overhead. CNN attempts to undersample the training data provided to KNN by determining a subset of the data that will result in the same classification accuracy, with less points needed, thus reducing the overall resources necessary to result in the same accuracy [2]. CNN is a greedy method so it does not guarantee to yield the optimal subset of data, but it has shown effectiveness in yielding similar (or sometimes improved) accuracies with smaller subsets of data [5]. CNN creates an empty set, *Z*, which is filled initially with 1 arbitrary point. The algorithm then loops through the training set randomly and grabs the 1-nearest-neighbor for the items in z. If its nearest neighbor is of the same class, it ignores it. If it is of a different class, it adds it to the Z set. This continues until Z is no longer changed. Z is then returned as a minimal subset of data that yields the same (or similar) accuracy as the initial set. Some pseudocode for CNN is below [2]:

```
$Z <- \emptyset$
$Repeat$
    $For\ all\ x \in X\ (in\ random\ order)$
        $Find\ x' \in Z\ such\ that\ \left||x - x'|\right| = min_{x^j \in Z}\left||x - x'|\right|$
        $If\ class(x) \neq class(x')\ add\ x\ to\ z$
$Until\ Z\ does\ not\ change$
```

C. K Cross Fold Validation

K Cross Fold Validation (CFV) is a commonly used statistical method that is applied across the machine learning field in order to more accurately measure algorithm performance. “K” in this instance is the number of folds to create. CFV creates several “folds” which are subsets of the overall dataset, and measures the performance of the machine learning algorithm(s) within that fold. Then, once you have performance scores for each fold, you can simply do a mean, distribution, or any other statistical measure technique to grade performance across all folds [3]. In the implementation of CFV for this experiment, there was an additional technique applied. The k number of folds were specified to be 5, and for classification usage, each fold must respect the distribution of classes initially presented in the original dataset. This was an additional layer of statistical protection designed to ensure folds did not inaccurately represent the classes.

IV. Experimental Design

A. Code Design

The project was written in Python, specifically utilizing the features of Python 3.7.3, and developed on a Windows 10 PC. The IDE of choice, for both ease and completion, was IntelliJ’s PyCharm.

B. Design of Experiment

The experiment was run with three test sets. Each test set performed was run multiple times, where the value of k is incremented from 1 to up to 20. The best classification accuracy or least error, and the k that caused that result, is then recorded for each run. The k values are stored in an array. If a given run’s performance is equal to the running best, that k value is recorded. If the current performance is superior to the running best, the k values array is reset, and that current k is added to the array. This ensures that all k values responsible for the most optimal results are recorded and published. Ultimately, the “best” values are reported via the terminal print outs.

Accuracy percentage was calculated using the following formula:

$$\left(\frac{\#\ correct\ attempts}{\#\ total\ attempts}\right) * 100$$

Error rates were calculated using mean squared error, or:

$$\frac{1}{N} * \sum_{i=1}^{N} * \ (A_i - \hat{A}_i)^2$$

The experiment then was run three different times, allowing for hundreds of randomized instances of KNN to run on the required datasets. The resulting accuracies, errors, and k were thus recorded and used for the analysis.

V. Description of Tuning Processes

A. Data Cleaning, Manipulation, and Handling

There was a fair amount of data cleaning work for this experiment. Those efforts are outlined below, grouped by dataset:

- **Ecoli Dataset:**
  - Deleted rows per project instructions
  - Changed Classifications to Numerical:
  - Formatted it to be opened as a csv
- **Segmentation Dataset:**
  - Deleted Rows 1-4 (just text / information about the dataset, no real data)
  - Moved Column A to be the last column (classification)
  - Changed Classifications to Numerical:
- **Forest Fires Dataset:**
  - Changed months to numerical
  - Changed days to numerical
  - Deleted column headings
- **Machine Dataset:**
  - Ignored non predictive features

Additionally, in various locations in my implementation, the datasets are randomly shuffled. This happens when data is initially read in from the csv file, so that each test set was not testing the exact same dataflow. Data is shuffled again before each fold is created in the CFV technique to reinforce the effort of not testing the same data in the same order. Other shuffles (random indices for CNN) are performed as well.

B. K Value for KNN

Most of the tuning for this assignment went into the adjustment of the K value for the KNN implementations. The K value for each run, classification and regression, starts at 1 where we simply observe the closest neighbor as determined by the Euclidean distance from the test point [to its closest neighbor]. Then, we continue to increase K to 20, ensuring that the most accurate (or the value with the smallest error) and its associated k are captured and reported.

The reason 20 was selected is that during development, it was noticed that the optimal k never exceeded 15, which is demonstrated in section VI and in the attached test runs. Additionally, once the code was finished and the experiment truly initiated, I found that there might be situations where the amount of nearest neighbors changes depending on which dataset and other tuning methods (CFV, CNN, etc.) are used, and that 20 was found to be a safe number of neighbors to assume every test point would have for either classification or regression, while still allowing for meaningful analysis.

VI. Results

The results of the three test sets can be examined below:

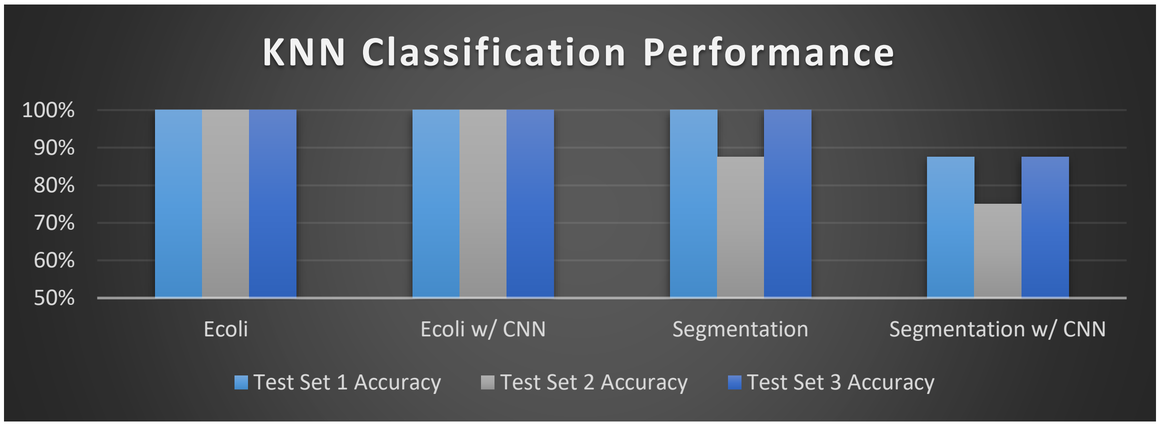
Classification Test Sets 1-3 Results

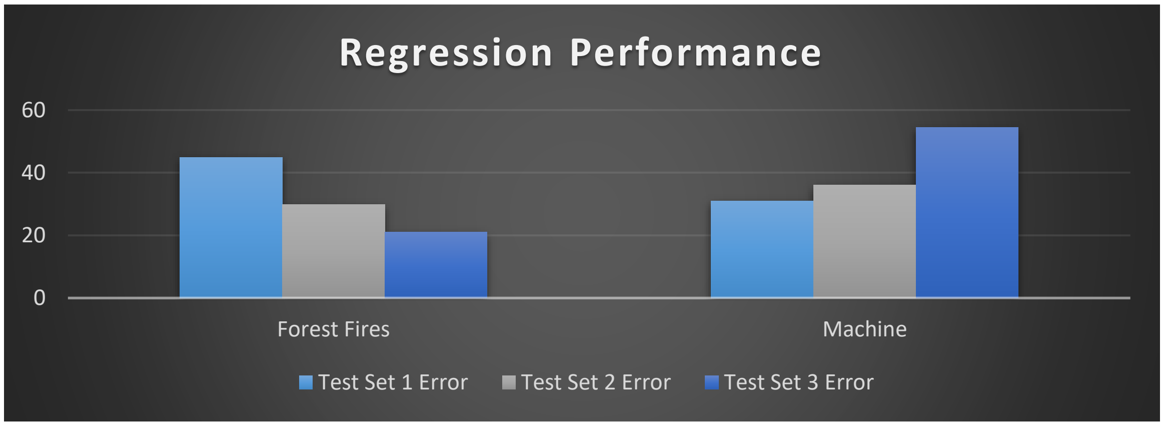
	Test Set 1 Accuracy	Test Set 1 K	Test Set 2 Accuracy	Test Set 2 K	Test Set 3 Accuracy	Test Set 3 K
Ecoli	100%	[1,2,3,4,5]	100%	[1,2]	100%	[1,2,3,4]
Ecoli w/ CNN	100%	[1,2,3,4]	100%	[1,2,3,4]	100%	[1,2,3,4]
Segmentation	100%	[1,2]	87.5%	[1,2,3]	100%	[1,2]
Segmentation w/ CNN	87.5%	[4]	75.0%	[1,2,3]	87.5%	[1,2]

Regression Test Sets 1-3 Results

	Test Set 1 Error	Test Set 1 K	Test Set 2 Error	Test Set 2 K	Test Set 3 Error	Test Set 3 K
Forest Fires	44.95	[6]	29.83	[3]	21.02	[7]
Machine	31	[1]	36.02	[3]	54.53	[2]

The following graphs were created from analyzing the above table:





VII. Analysis and Conclusion

Conclusively, I felt like this was a very useful exercise and probably the most rewarding thus far. The implementation of KNN and CNN really helped to solidify my understanding of non-parametric learning models and helped me to better grasp what it means to not make any assumptions about the data with which we are working, and the levels of flexibility that provides (i.e., using the same algorithm for classification and regression). Additionally, the requirement to implement CFV was well received, since there is immense benefit towards using CFV in all future assignments to have a much better statistical analysis of my results.

The results of this experiment were surprising. I was not anticipating such a small number of neighbors would result in such positive results, especially for classification. Except for one classification data point, k-neighbors under 5 yielded the best results. The results were overwhelmingly positive (nearly 100%

classification), which means that the model was running the risk of overfitting. However, being run on 5 separate folds and across three test sets would lead me to believe that the model is not overfitting, and instead, the algorithm is just performing as it is designed to.

For the regression datasets I was anticipating a smaller number of neighbors to be used, since we are taking the average of a series of values and calculating an error. However, it was surprising to me that the average for the regression sets was about 3.67 neighbors, which is lower than I was initially suspecting in my hypothesis. Additionally, the mean squared error proved to be relatively low, which again came as a surprise. I suspect this to be because the algorithm correctly gathered the proper number of nearest neighbors, and there was a lot of attention given towards reporting the optimal error (i.e., the smallest error for a single k value across 5 folds). I was not surprised to see that for any of the regression instances there were no 2 k values which reported the exact same performance; this is expected, as we are taking a mean of the values of neighboring points which would (hopefully) be different for every run.

Despite a lot of successes with this assignment, there are several potential shortcomings. The first and foremost was the usage of the Euclidean distance function as outlined above, which does not work well in high dimensional spaces. Therefore, it could be possible that the Euclidean distance function was miscalculating its “closest” neighbors and thus gathering erroneous results. Additionally, though CFV was used to ensure good statistical processes, there still could have been some feature selection techniques applied to try to improve model performance. Lastly, and importantly, it should be noted that I am by no means an expert software developer, and that I have historically had difficulty implementing the algorithms from scratch in this class (even with an understanding of their functionality). It is very possible that this led me to make logical errors in my code, or even in my data handling.

Regardless, I am still very pleased with the results of this experiment and feel that this was the most rewarding project implementation yet, due largely to how well I felt I understood the material and how this integration effort resulted in very positive results. Even though there is room to improve, I thought this assignment was extremely useful and rewarding.

VIII. References

[1] C. G. Dheeru Dua, "UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]," University of California, School of Information and Computer Science, Irvine, 2019.

[2] A. More, "Survey of resampling techniques for improving classification performance in unbalanced datasets," Cornell University, 2016.

[3] E. O. J. Laaksonen, "Classification with learning k-nearest neighbors," ICNN'96, Washington, D.C., 1996.

[4] E. Alpaydin, Introduction to Machine Learning 3rd Edition, Massachusetts Institute of Technology, 2014.

[5] E. Alpaydin, "Voting over Multiple Condensed Nearest Neighbors," Artificial Intelligence Review, vol. 11, 1999.

[6] A. P. J. A. L. Juan D. Rodriguez, "Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 3, pp. 569-575, 2010.