Journal of Machine Learning Research 1 (2020) 1-7 Submitted 9/23; Published 9/27

K-Nearest Neighbor Implementations

Brian Loughran BLOUGHRAN618@GMAIL.COM
Department of Machine Learning
Johns Hopkins University
Baltimore, MA 21218, USA

Editor: Brian Loughran

Abstract

This document describes the k-nearest neighbor algorithm as a method for creating a model to predict real-world data sets. Included in this paper is a problem statement which includes assumptions made for the algorithm and projections on how it is expected to perform against a variety of different data sets. Also discussed is a description of the experimental approach, as well as any processing that has to be done on the input data to fit in the k-nearest neighbor framework, as well as some methods for reducing the size of the data model. Results for the algorithm are presented for multiple data sets, as well as some modifications that can be made to the algorithm to improve performance/compute time. Finally, conclusions are drawn for performance.

1 Problem statement

K-nearest neighbor is a supervised learning algorithm which can be used to solve both classification and regression problems. The algorithm works of the basic assumption that similar things exist in close proximity to each other. A multitude of data sets are considered to test the k-nearest neighbor algorithm including regression models abalone, forestfires, and machine, as well as classification models glass, house-votes-84, and segmentation. Predictions are made based on the k-nearest neighbors, and performance for k-nearest neighbors algorithm is done by comparing the predictions to the given result in the data set. The algorithm is not a linear classifier, thus can learn more complex models than linear classification problems effectively.

For evaluating algorithm performance we use a 5-fold cross validation. 10% of the data is taken out before the 5-fold cross validation for optimizing hyperparameters. The remainder of the data is used for the 5-fold cross validation, and performance over each of the folds is considered to get the performance over the entire set.

There are several factors that come into play when considering how well k-nearest neighbor will perform. Since we are tuning hyperparameters such as k, σ, and ε, we will exclude the effects of these hyperparameters in this section. However, other factors, such as the amount of data present in the data model will effect performance. Increasing the data set size will increase model accuracy in general, but will have the effect of diminishing returns as the data set increases, as well as increase compute time for a given point. Overlap between neighbors will also play a role in how well k-nearest neighbors performs. Sets where different classes are clearly separated will be classified quite easily by k-nearest neighbor, while sets with classes that overlap greatly will be more of a challenge. It also may be difficult to classify against sets that have very few members of a given class if another class with more data points begins to encroach or overlap the class with fewer points.

Using the points described above we can begin to predict which data sets will give good performance for nearest neighbor, and which classes are expected to perform less well. We will begin with the regression sets: abalone, forestfires and machine. Abalone is expected to perform very well due to the high amount of data in the set and low overlap between different age ranges.

Forestfires is expected to have more of a challenge since the numbers of acres burned can vary so greatly and is not expected to follow a normal distribution. Machine is expected to perform decently due to average-level overlap between different price ranges. Next we can discuss the classification sets: glass, house-votes-84, and segmentation. It is expected that glass will perform not well due to the low number of test points in the data set and the high overlap between classes. house-votes-84 is expected to perform very well, since different political parties have different voting patterns. And segmentation is expected to perform ok; while there is nothing concerning about the size of the data set and the classes, k-nearest neighbor is not expected to have a bias that matches this problem particularly well.

Also included is a discussion of some ways to limit the size of the training set, namely edited nearest neighbor and condensed nearest neighbor. The goal of edited nearest neighbor is to edit out points that are incorrectly classified with the goal of reducing the size of the training set and decreasing noise in the training set. Condensed nearest neighbor has a similar goal of reducing the size of the set, but does this by adding points that classified incorrectly to the data model until all points in the data model would be correctly classified by the subset. Each of these methods is expected to reduce the total compute time for k-nearest neighbor, and provide similar accuracy levels to the entire set.

2 Experimental Approach

K-nearest neighbor is a well-known algorithm, thus the algorithm will not be discussed in this section. Some assumptions are made throughout the k-nearest neighbor logic that are discussed in this section, including reading in data sets and assumptions made in the computations as well as implementation details.

The first thing that has to be determined for k-nearest neighbor is the computation of the distance function. The distance function can be computed using Manhattan distance, Euclidian distance, or a higher order distance function. Comparisons for each are not included in this report, however it should be noted that Euclidian distance is used for the distance calculations for all sets.

Next, we note that the values for k-nearest neighbor must all be numeric. The one exception is for the classification sets, the result vectors may be numeric or string, however the result vectors for regression must be numeric as well. This leads to some preprocessing that has to be done for some of the data sets. For example, in the abalone set, the sex is either male, female, or infant. This value was one-hot-coded, thus giving a distance of 0 if the sex agrees, and 1 if the sex does not for that parameter. There are more sophisticate ways of dealing with this, for example feature difference matrices, however for this case and this data set, looking at the distance data and regression performance, this was deemed to be overkill for this implementation. For forest fires, the month and day values are strings, and were encoded to numeric values (Jan=1, Feb=2, etc.). While this is an effective way to compute the distance between Wednesday and Friday, it can produce weird results for computing the distance between Sunday and Monday (Sunday=7, Monday=1). While feature difference matrices were also considered for this, again looking at the regression performance and distance calculations it was deemed that the feature difference matrix would produce negligible performance improvements. Similar to sex for abalone, votes on the house-votes-84 set needed to be converted from [y, n, ?] to numeric, and this was done using one-hot encoding, again with the determination that feature difference matrices would produce negligible improvements in performance. For the machine data set, some attributes needed to be dropped, including the model since it was mostly irrelevant, and the PRP because that was a published linear regression estimate, thus would have biased our predictions. Vendor value was one-hot-encoded which results in a distance of 0 if the vendor matches and 1 if the vendor does not match. Some rows needed to be skipped at the head of the segmentation data set to read into the standardized format.

Another consideration for classification data sets is ensuring each of the sets generated (tuning set and 5 validation groups) has a representative sample from each class. The way that this was done was to start by ordering the data set as a whole based on the result. The tuning set then took every 10th data point from the set, thus resulting in a representative sample for the tuning set. Each of the validation groups took every 5th data point from the remaining set resulting in each of the validation sets also having a representative sample.

Because each of the data sets require di_erent decisions to be made for processing the data sets (which values to one-hot-code, which values to discretize, whether to re-order the data, missing attributes etc.) an options _le is created for each data set with a .options.yaml extension. This file specifies each of the decisions that has to be made for each of the data sets and automates the preprocessing of the sets.

Some hyperparameters are tuned for both classification and regression; however the hyperparameters are not the same for each. Classification is a simpler case, since the only hyperparameter that has to be tuned is the number of neighbors to consider (k), and the loss function to be optimized is 0/1 loss (whether the classification was correct or not). Regression is a bit more nuanced. The tuning for regression starts by tuning k and σ (Gaussian kernel range). k and σ values are optimized using a root mean squared method, similar to what is used in linear regression. After k and σ parameters are optimized to best fit the data, another parameter, ε is tuned. ε represents the acceptable error band for a regression guess, and this value is tuned to increase until the performance does not improve further. While this will result in increasing ε while there is still error in the tuning set, examining the value of ε after the regression is run can indicate how accurate the regression was.

Edited nearest neighbor consists of the simple process of removing points from the data set that are deemed not necessary. While one can choose between points that are incorrectly classified or points that are correctly classified, this implementation considers points that are incorrectly classified in an effort to remove noise from the system. This implementation uses a batch process to edit neighbors out. There are two ways to successfully terminate the editing process. The first is if a batch edit does not remove any more neighbors. The other is to check the performance of the edited set against the performance of the unedited set against a separate tuning set. If the performance degrades, the unedited set is returned, while the editing process will continue if performance improves or stays steady. This can result in an edited set having the same number of points as an unedited set if the performance immediately degrades.

Some discussion should be devoted to the implementation of condensed nearest neighbor. Since the hyperparameters are not tuned for this process, $k=1$ and $\sigma=1$ are used (noting that σ value cancel itself out in the case that $k=1$). ε is defined as 1% of the result range to try to get a good fit for the regression case. One suggestion made for ordering the points to be considered by the condense process was to have the points with the least total distance to other points first, and points with greatest total distance last. In testing (which will be discussed further in the results section), it was found that the condensing process was very computationally expensive. Considering points by distance first would add significant compute time, thus the points were considered in the order listed in the data set (so effectively random order). The condensed set is returned once each of the points in the test set are correctly classified by the condensed set.
One last implementation detail to discuss is in the Gaussian kernel implementation. I found that while tuning sigma, for very small sigma values and very sparsely populated neighbors, I may get a ZeroDivisionError. This has to do with the way that python handles float values, and how very small numbers round to 0. To avoid this, in the case of the ZeroDivisionError, a plurality vote is employed instead. This was an edge case that did not occur often, thus it is expected to have negligible impact on predictive performance.

3 Results

K-nearest neighbor was run on 3 regression data sets (abalone, forestfires, machine) and 3 classification sets (glass, house-votes-84, segmentation). Condensed and edited k-nearest neighbor were also run for each of the 6 total sets, resulting in 18 total runs. In running the algorithms, it was determined that since the output was so verbose, it would make sense to include a output.txt file as well as an output summary.txt file, for viewing either general algorithm performance or detailed line-by-line values requested in the assignment. The name of the output file is of the format set.output.txt or set.summary.txt, depending on whether you want the full output or the summary, and where set is the set name. Edited or condensed nearest neighbor output files are denoted by set-edited.output.txt or set-condensed.output.txt. Thus there are 6 different output files per set (output/summary, -edited.output/ssummary, -condensed.output/summary), resulting in a total of 36 different output files.

There is a wealth of information included in the output files, and referencing those files should give greater insight into low-level items not included in this report. What is included in the *output.txt files is the following:

- Demonstration of mapping non-binary categorical variables with one-hot coding
- The preprocessed pandas dataframe corresponding to the choices laid out during the experimental approach section
- Demonstration of splitting data into a tuning set with 10% of the data and 5 cross validation groups with 18% of the data each
- For each test points considered, a summary of the point and it's parameters
- For each test point considered, the index of the nearest neighbor in the training set as well as their distance and a summary of the training points.
- For each test point considered, the guess based on the nearest neighbor and their distance and the kernel function for regression or the plurality vote for classification
- For each test point considered, the actual result
- Any points removed from the edited nearest neighbor set
- Any points added to the condensed nearest neighbor set
- The severity of the data reduction for each of the 5 cross validated training sets if either edited nearest neighbor or condensed nearest neighbor was called
- The model, including k, σ, and ε (if applicable) for a tuned cross-validation
- The performance of k-nearest neighbor for each of the 5 cross validations
- The average performance of the k-nearest neighbor algorithm across all 5 folds

Note that one considerable omission from the output.txt file is a demonstration of the distance function. In testing it was found that adding the calculation of the distance function increased the file size nonsensically, as well as reduced file readability considerably. Output lines are included in the distance function (commented) if lower level debugging needs to be done or demonstrated.

There is significantly less information in the *summary.txt file, making it a place to go for high level information about the algorithm performance. The only items included in the *summary.txt file are the following items:

- The severity of the data reduction for each of the 5 cross validated training sets if either edited nearest neighbor or condensed nearest neighbor was called
- The model, including k, σ, and ε (if applicable) for a tuned cross-validation
- The performance of k-nearest neighbor for each of the 5 cross validations
- The average performance of the k-nearest neighbor algorithm across all 5 folds

Below is a summary of the performance for each of the data sets, grouped by regression and classification sets. One item to note is that we do not have condensed performance for either the abalone or forestfires data sets. While implementing edited and condensed nearest neighbor reduces the time to make an estimation for a given point, the amount of time to edit or condense the data sets is significantly more than the benefits received from making a guess with the reduced model size. This results in greater algorithm execution time for edited nearest neighbor and condensed nearest neighbor in relation to regular nearest neighbor. The compute time was so greatly impacted for larger data sets like abalone and forestfires, the algorithm was not able to complete in time for this report (~20% completion after 20 hours of compute time on a 2014 Lenovo Y50P).

Regression			
Set	k-nearest Performance	Edited Performance	Condensed Performance
abalone	100%	100%	-
forestfires	98.28%	99.35%	-
machine	97.87%	97.34%	98.4%
Classification			
Set	k-nearest Performance	Edited Performance	Condensed Performance
glass	67.19%	65.62%	66.67%
house-votes-84	92.33%	92.07%	91.82%
segmentation	78.31%	77.78%	77.25%

Table1: Performance Summary

Clearly the performance of the regression sets were very good. However, because of the way that ε was tuned, given that ε would continue to increase while performance increased, it is potentially more interesting to look at the value for ε to determine accuracy. An accuracy of 100% is not particularly impressive if the ε value is a high number. Thus examining ε can give us a more full picture into the accuracy of the nearest neighbor classification problem for the given set. ε values are tabulated below for the regression sets (of course they would have no meaning for the classification sets:

Set	k-nearest ε	Edited ε	Condensed ε
abalone	25	25	-
forestfires	250	250	-
machine	250	250	250

Table 2: ε values for regression sets

While the accuracy numbers for the regression sets are impressive, the ε bands are more telling of the performance. For example, this would indicate that abalone is 100% accurate within an age range of 25, which is less impressive considering the entire age range of 29 years. Forestfires is ~98% accurate within 250 acres of land, with the range of burned land being 1090 acres. Machine is ~97% accurate within 250 of price with maximum price being ~1000. Narrower ε bands can be specified to gauge closer accuracy, however performance will always decrease as ε decreases.

4 Algorithm Behavior

K-nearest neighbor is an interesting algorithm considering that there is no true “training” phase, since the model “training” is just recording all the points in the training set. All of the computation (with the exception of the edited and condensed cases) come during the testing phase, which is in contrast to other algorithms like naïve-bayes, winnow, radial basis function, etc.

Discussed previously is the effect of increasing and decreasing the value of ε for regression problems. Another indicator for the accuracy of the regression is to compute the value of ε necessary to correctly classify n% of the test set (selecting a value for n like 50%, 75% or 90%). This would tell you how tight the regression is for a given data set, and give you the ability to compare across different regressions to determine which is the tightest fitting based on a single number, since the combination of accuracy and ε can prove misleading for some data sets.

Both edited nearest neighbor and condensed nearest neighbor do a decent job of reducing the size of the test set. However, the process of editing and condensing also take up a large amount of compute time, more than their benefit, resulting in edited and condensed nearest neighbor taking longer than simple k-nearest neighbor to run. While it is true that editing and condensing the training set will reduce compute time for each point tested, unless the points are saved externally before the nearest neighbor process, this will not result in reduced compute time for the entire process.

It is also interesting to note that there were essentially negligible performance differences between k-nearest neighbor, edited nearest neighbor and condensed nearest neighbor. This is partially by design, but edited nearest neighbor typically reduced the training set by 20-50%, and the condensed nearest neighbor typically reduced the training set by 50-75%, thus the negligible loss in performance was notable.

5 Conclusion

As discussed in the problem statement, k-nearest neighbor is not a linear classifier, thus should be able to perform quite well with non-linear problems. It was predicted that k-nearest neighbors would perform well on abalone and house-votes-84, decently on machine and segmentation, and less well on forestfires and glass.

The performance was basically as expected for all the data sets. While the ε value for abalone was relatively high, the regression loss was relatively low and the guesses pretty good, the high ε value is likely due to the greedy tuning of ε. There is a similar story to be told for both forestfires and machine. The performance of glass, house-votes-84 and segmentation were predicted in the order shown in results, so the understanding proposed in the problem statement of the report would likely hold up well. As an aside, I think it may be interesting to see the house voting patterns for 2016/2020 to compare to house-votess-84. I would compare the accuracy between the two models to see if the political climate of today was more polarized than in 1984.

For data sets where there was data on naïve-bayes and winnow-2 algorithm (house-votes-84 and glass) it was found that k-nearest-neighbor performed better than either winnow or naïve-bayes for both implementations. While the performance was better, it was not by too significant an amount (~2%), so while there is certainly a performance boost, it is not massive.