Introduction to Machine Learning

CSCE 478/878

Programming Assignment 1

Fall 2020

K Nearest Neighbors Model for Binary Classification

Part A (Model Code): 478 (50 pts) & 878 (65 pts)
Part B (Data Processing): 478 (20 pts) & 878 (25 pts)
Part C (Model Evaluation): 478 (40 pts) & 878 (50 pts)
Part D (Written Report): 478 & 878 (30 pts)

Total: 478 (140 pts) & 878 (170 pts)

Last Name 1: Adamec

First Name 1: Matous

NUID 1: 50018756

Last Name 2:

First Name 2:

NUID 2:

Last Name 3:

First Name 3:

NUID 3:

Obtained Score:

K-Nearest-Neighbor Brute Force Algorithm Optimization For Wine Quality Classification

Matous Adamec
University of Nebraska-Lincoln
Email: matous.adamec@huskers.unl.edu
Date: September 10, 2020

***Abstract*—The brute force K-Nearest-Neighbor (*KNN*) algorithm is implemented to classify the quality of wine based on a dataset containing 11 features and nearly 5000 instances. The algorithm is evaluated and compared on several parameters, including k, distance metric, weight type, and acceptance threshold, using 4-Fold cross-validation in the fitting stage. We find that the optimal implementation of the KNN algorithm over the wine dataset has k = 1, Manhattan distance metric, and uniform weighting, without any dependence upon the acceptance threshold due to the constraint of k = 1.**

1. Introduction

KNN is a machine learning algorithm which fits a set of training data and then makes predictions based on new data's proximity to old fitted data. This proximity may be measured through various vector norms, have different styles of weighting, and possess different thresholds of acceptance. Other parameters may also vary, most relevantly the number of nearest neighbors (K in the algorithm name). The goal is to "tune" these variables and find the most efficient subset for our given dataset, which in this case is a relatively small analysis of wine quality.

2. Data Summary

The dataset focuses on white wines, featuring 12 different variables. Of those 12, we are particularly interested in evaluating wine quality, which is assigned a number according to the other 11. For the purposes of our analysis, we convert this quality variable into a binary format: we consider it good if quality > 5 and bad if quality ≤ 5. This binary quality variable shall be our algorithm's target. A summary of all variables, and their basic statistics, is in Table 2.

In practice, some of these variables have wide-ranging differences in variance, which leads to certain features having more weight than others when the algorithm is fitted. To remedy this issue, we standardize our data by modeling it as approximately Gaussian, and thus centered around its mean. Then we can normalize the data by dividing each features' variance, getting us unit variance and eliminating these discrepancies. In other words, for each value (x) of every feature (f), we find the mean and variance of the feature (x_f and σ_f), and apply the transformation $\frac{x-x_f}{\sigma_f}$. This helps the algorithm consider all features equally, getting us a better set of fitted results to make predictions off of.

TABLE 1. Effects of Data Standardization with k=5, Euclidean Metric, and Uniform Weighting

	Unst. Accuracy	Unst. F1	St. Accuracy	St. F1
Run 1	0.6289	0.7313	0.7110	0.7910
Run 2	0.6468	0.7511	0.7138	0.7933
Run 3	0.6483	0.7442	0.7210	0.7980
Average	0.6413	0.7422	0.7153	0.7941

The algorithm was ran three times with and without standardization, and the results on test data are shown in Table 1. Clearly, the standardized results are better in both accuracy and F1 score. We show less clear results between standardized unweighted data and standardized inverse-distance weighted data in Table 3, which shows that unweighted data has lower accuracy but higher F1 score. However, the difference between accuracy was higher than that of F1 score.

There was also some redundancy in our data, stemming from highly-correlated features. If two features are highly-correlated, then evaluation on one is identical to evaluation on the other, and is simply wasting processing time. Worse, it'll be counted doubly in any distance metric that we apply, effectively making that characteristic of the data worth twice as much as any other when new data is classified. A grid of correlations for this dataset is demonstrated in Appendix Fig. 1.

I made the choice to discard features which had correlation coefficients $|r| > 0.4$. In particular, I had found that *Total Sulfur Dioxide* correlated highly with several other features, namely *Residual Sugar*, *Free Sulfur Dioxide*,

TABLE 2. SUMMARY STATISTICS OF ALL VARIABLES IN WINE DATASET

Index	Fixed Acid.	Vol. Acid.	Citric Acid	Res. Sugar	Chlorides	Free SO_2	Total SO_2	Density	pH	Sulphates	Alcohol	Quality
Count	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898
Mean	6.85479	0.278241	0.334192	6.39141	0.0457724	35.3081	138.361	0.994027	3.18827	0.489847	10.5143	5.87791
StDev	0.843868	0.100795	0.12102	5.07206	0.021848	17.0071	42.4981	0.00299091	0.151001	0.114126	1.23062	0.885639
Min	3.8	.09	0	0.6	0.009	2	9	0.98711	2.72	0.22	8	3
25%	6.3	0.21	0.27	1.7	0.036	23	108	0.991723	3.09	0.41	9.5	5
50%	6.8	0.26	0.32	5.2	0.043	34	134	0.99374	3.18	0.47	10.4	6
75%	7.3	0.32	0.39	9.9	0.05	46	167	0.9961	3.28	0.55	11.4	6
Max	14.2	1.1	1.66	65.8	0.346	289	440	1.03898	3.82	1.08	14.2	9

TABLE 3. EFFECTS OF WEIGHTING (UNIFORM VS. INVERSE-DISTANCE) WITH K=5, EUCLIDEAN METRIC

	Un. Accuracy	Un. F1	I-D Accuracy	I-D F1
Run 1	0.7110	0.7910	0.7394	0.7893
Run 2	0.7138	0.7933	0.7171	0.7757
Run 3	0.7210	0.7980	0.7369	0.7849
Average	0.7153	0.7941	0.7311	0.7833

Density, and *Alcohol*. It thus seemed prudent to remove *Total Sulfur Dioxide*, as these other features did not necessarily correlate heavily with each other in as obvious of a way. *Alcohol* was also found to correlate heavily with *Residual Sugar*, *Density*, and *Residual Sugar*, and dropped from consideration as a result. Finally *Residual Sugar* was dropped because it had a very high correlation with *Density*.

It is useful to check that we did not drop any significant information as a result of this. For example, we dropped *Residual Sugar* at the end, but it was also correlated with dropped features *Total Sulfur Dioxide* and *Alcohol*. I would argue that this is not important: in the former case, we kept *Free Sulfur Dioxide*, which preserves the information from *Total Sulfur Dioxide* (r = 0.62) even if *Residual Sugar* is dropped; in the latter case, we kept *Density* which preserves the information from *Alcohol* (r = -0.78). We can similarly show that stand-ins remain for both *Total Sulfur Dioxide* and *Alcohol*, the other two features we dropped.

3. Methods

The brute-forced implementation of our KNN algorithm makes predictions with O(mnw) complexity, given m features, n instances of data, and w trained points. This is because it iterates through each input row of data (n operations), making w comparisons with trained data across m features (w * m operations). If we assume that the number of features is significantly smaller than the number of instances (which *should* be the case in big data), then our algorithm tends towards O(nw), which we can estimate as O(n^2) in the case of a 50/50 train/test split (which is probably close to the worst case - we want a lot of training data!) In either case, this simplification comes with very large constants to our O(n^2), so our algorithm would perform especially poorly for small data.

The value of k does not impact our run-time performance. We evaluate the distance between a test point and all training points whenever we make a prediction, and selecting the k-smallest values is a minor exercise in small-order statistics, which can be accomplished in O(n) time. Selection of k, does, however, have an impact on the efficiency of our prediction algorithm. Specifically, it determines how many of the smallest values we poll during classification. When k = 1, we assign a value based on the closest neighbor, when k = 3, we poll the three closest neighbors, and so on. Increasing k can be a good way to reduce the impact of anomalies in the training data (since a single point cannot dictate classification). Making k too high, however, can be bad: at the extreme, if k is the size of the dataset, then any newly-added point is just being classified by the majority, rather than any proximity.

The choice of metric and weighting similarly do not impact run-time performance, as the former is a difference in calculation without a change in the number of points being calculated, and the latter is a simple multiplication factor in the polling step. They may, however, impact performance. The choice of higher-order metrics (such as Euclidean) results in dissimilar dimensions having different weights, since we are squaring the distances in each dimension, which can be problematic. It can, however, be a better representative of distance for similar data dimensions, and thus choice of metric depends heavily on the type of data you are working with. Similarly, weighting makes closer values have a greater input when neighbors are polled, which can emphasize anomalies but provide more intelligent classifications. Of particular interest is the relationship of k and weighting; for example, higher k may be used to reduce anomalies in combination with distance weighting to achieve a perfect balance for classification. As we will see in the results, we chose not to use inverse-distance weighting because it led to terrible F1 Scores when the algorithm was tested.

4. Results

A full list of performance results for all combinations of parameters can be seen in Table 4, which is ordered from best to worst. In addition, the best k-value for each pair of metric and weights is provided in bold, to emphasize the best combination of that pair and to highlight the relative consistency of the best k-value around k = 1.

From these choices, we can see that the best performance

TABLE 4. ORDERED F1 SCORES OF PARAMETER COMBINATIONS

Parameter	F1 Score
k=1, Manhattan, Uniform	**0.8249**
k=9, Manhattan, Uniform	0.8231
k=11, Manhattan, Uniform	0.8225
k=5, Manhattan, Uniform	0.8163
k=1, Euclidean, Uniform	**0.8053**
k=11, Euclidean, Uniform	0.8029
k=9, Euclidean, Uniform	0.7995
k=1, Euclidean, Inverse-Distance	**0.7940**
k=5, Euclidean, Uniform	0.7879
k=1, Manhattan, Inverse-Distance	**0.7177**
k=5, Euclidean, Inverse-Distance	0.6565
k=9, Euclidean, Inverse-Distance	0.6093
k=11, Euclidean, Inverse-Distance	0.5865
k=5, Manhattan, Inverse-Distance	0.4241
k=9, Manhattan, Inverse-Distance	0.2691
k=11, Manhattan, Inverse-Distance	0.2265

(measured by F1 Score) is found with the k = 1 model that uses the Manhattan metric and uniform weighting. This is the model which we will use from here on out, to dive deeper into some performance metrics and optimization as we employ it on our test data.

Deploying our *(k=1, Manhattan, Uniform)* model on test data, we can evaluate broader performance metrics, which are detailed in Table 5. The performance of our model in precision and recall is correlated with its F1 Score (which is expected), and all three show fairly good performance. Our accuracy is 0.8335, which could be better (we're misclassifying about 17% of data!), but KNN is a rather naive machine learning algorithm. The 95% confidence interval on our generalization error is 0.1430 $\leq$ GE $\leq$ 0.1900. Our confusion matrix shows that we misidentify 84 as false negatives, and 77 as false positives, which is a significant amount out of the total ($\tilde{1}000$). In Figure 1, we plot the ROC and Precision-Recall Curves for our model. Note that since k = 1, our acceptance threshold has minimal impact, because the nearest neighbor will always make up 100% of the poll. For comparison, I also evaluated the ROC and Precision-Recall curves for the *(k=9, Manhattan, Uniform)* model, which came in at a close second with F1 score. The results of this second model are shown in Figure 2.

The optimal value in our ROC curve, since we have no preference for TPR or FPR in this problem, is the inflection point of the curve (minimizing FPR, maximizing TPR). In Figure 1, we can see that the optimal threshold occurs when TPR is about 0.9, and FPR about 0.2. This corresponds to any threshold, t, $0 < t \leq 1$. Again, this is because with only one neighbor, we're always getting 100% in the poll of neighbors, so any threshold that is able to be overcome (i.e., $\leq$ 1) will behave the same. We are

TABLE 5. PERFORMANCE OF *(k=1, Manhattan, Uniform)* MODEL

Metric	Performance
Precision	0.8810
Recall	0.8716
F1 Score	0.8762
# of TP	570
# of FP	77
# of FN	84
# of TN	236
Confusion Matrix	$\begin{bmatrix} 570 & 77 \\ 84 & 236 \end{bmatrix}$
Accuracy	0.8335
Generalization Error	0.1665

thus free to use the precision-recall curve to determine the optimal threshold, which is where an intersection occurs. Again, this is any threshold $0 < t \leq 1$. Thus, choice of threshold doesn't matter in the case of our first model.

In Figure 2, the inflection point occurs somewhere around TPR = 0.5, FPR = 0.1. The threshold point corresponding to this is 0.8, and thus the optimal point for the second model's ROC is 0.8. On the other hand, for the precision-recall curve, the optimal value is around 0.6, where the two precision and recall curves intersect.

5. Conclusions

Our results indicate that the optimum KNN classifier model on the wine quality dataset is one in which k = 1, using the Manhattan metric for measuring distance, and uniform weighting. In general, the Manhattan metric and uniform weighting perform best across all values of k tested (k $\in$ [1, 5, 9, 11]). Uniform weighting is the more important feature, however, as it outperforms inverse-distance across the board (except when k = 5 and Euclidean distance is used, but it is only marginally behind the *(k = 1 Euclidean inverse-distance)* model).

The optimal threshold for the best-performing KNN classifier is not relevant, as it performs well on all thresholds which are not extrema, since it only polls a single point when determining classification. For the second-best KNN classifier, the optimal threshold for the ROC curve occurred at t = 0.8, and the optimal threshold for the precision-recall curve occurred at t = 0.6. This discrepancy in thresholds might be further optimized, and thus we can safely say that $0.6 \leq t \leq 0.8$, with some weighting applied depending on the importance of the ROC curve or precision-recall curve to our data. This would be a complex exercise, as the ROC and precision-recall curves change at different rates, and the direct impact on performance of sub-optimal thresholds is not immediately clear for either. Further evaluation would be necessary to determine this t-value.

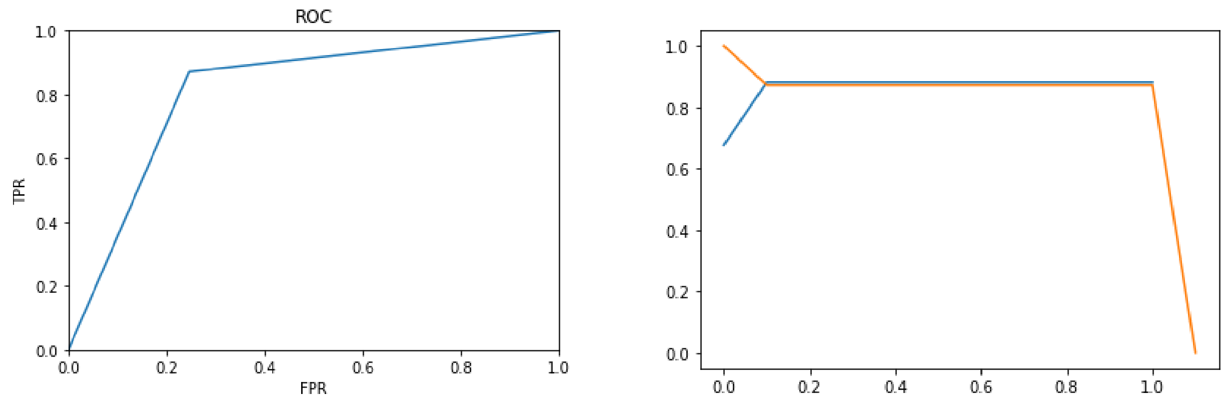

Figure 1. ROC (AUC = 0.8128) and PRC curve for *(k=1, Manhattan, Uniform)* model.

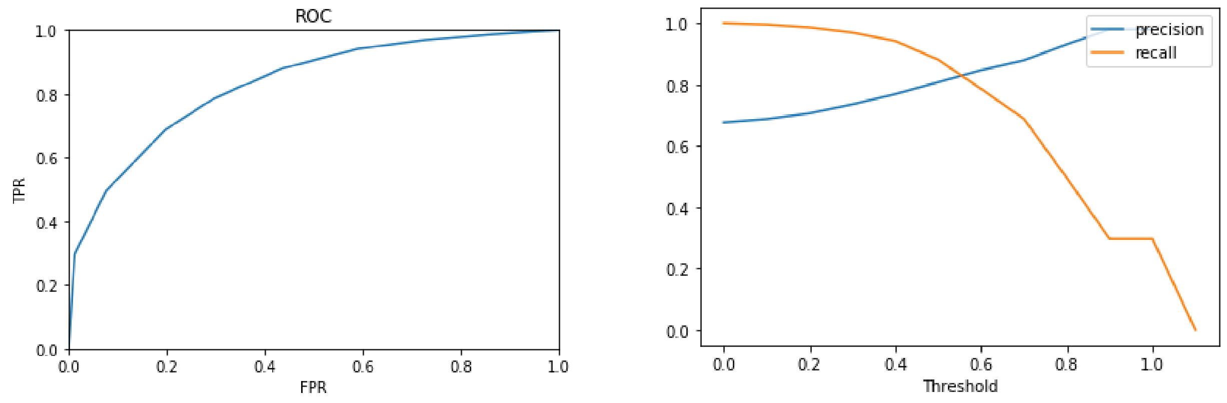

Figure 2. ROC (AUC = 0.7738) and PRC curve for *(k=9, Manhattan, Uniform)* model.

Appendix

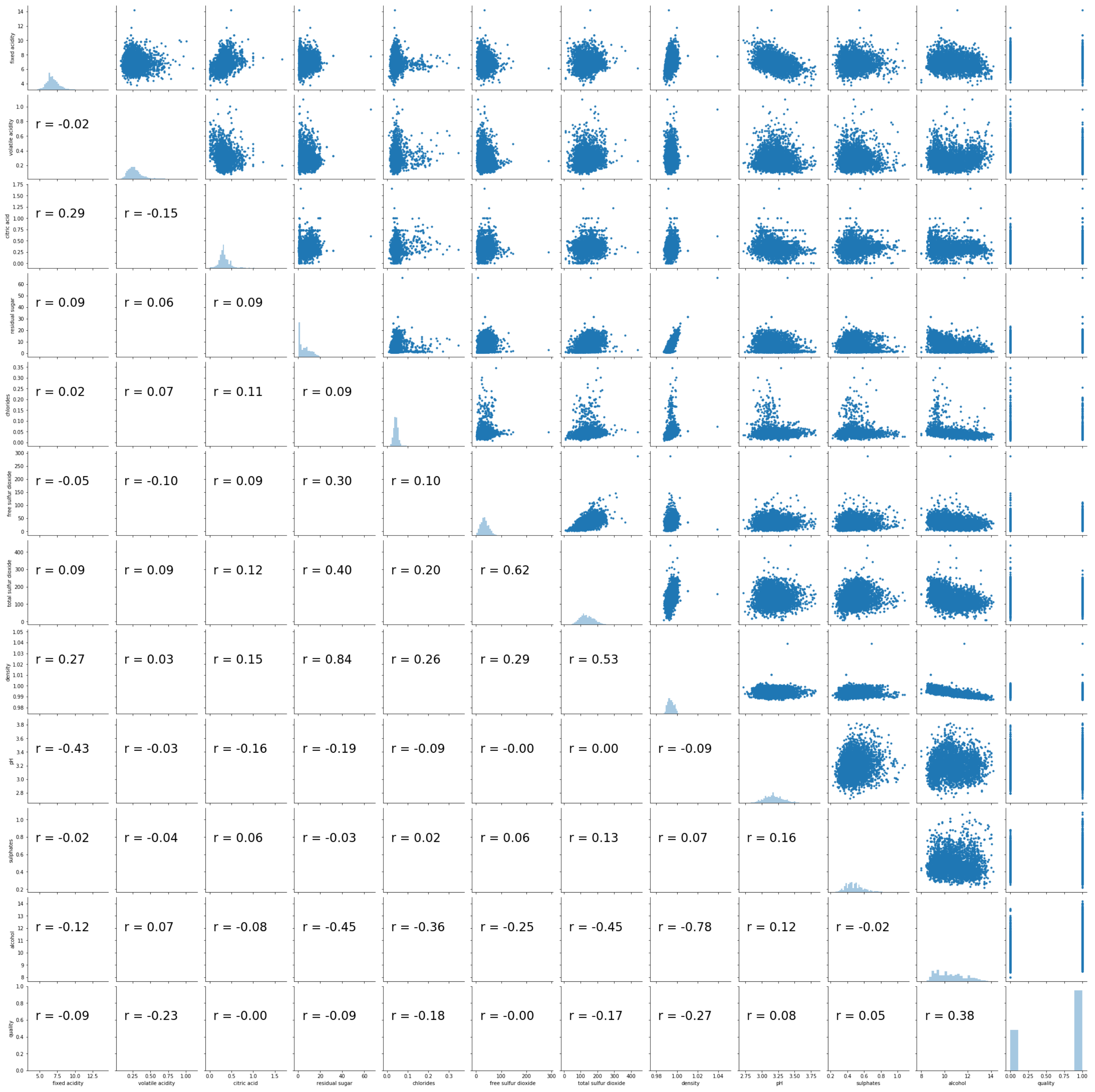

Figure A.1. Correlation plots for features of the wine quality dataset.