k-NN Classifier Performance on Wine Quality Dataset

Rabindra Nepal

Department of Physics, University of Nebraska-Lincoln

Feb 7, 2019

Introduction

In this brief report, we discuss the performance of an in-house brute-force k-nearest neighbors (*k*-NN) classifier for a binary classification problem on wine quality dataset. Note that all the required data preprocessing and model performance-measures functions used in this work are also self-implemented. More specifically, we only use python NumPy package and don't use any machine learning libraries. With our in-house version of brute-force *k*-NN model, we achieve the maximum F1-score of 0.890 on the considered dataset.

Dataset

The dataset used is taken from UCI Machine Learning Repository for 4898 Portuguese wine samples (Cortez, 2009). This dataset was originally used to study the human wine taste preferences using machine learning algorithms such as support vector machine, multiple regression and neural network (Paulo Cortez, 2009). The dataset has twelve different features including the overall quality measure in a scale of 0-10. The dataset is complete and there are not any missing values. All the feature values are in float64 except the quality feature which is given as an int8. We use 'quality' feature as the target in this study and the classifier is used to classify the wine samples into two binary categories: 'good' for $0 \leq \text{'quality'} \leq 5$ and 'bad' for $5 < \text{'quality'} \leq 10$ based on the rest of the features.

Table. [1]: Statistics of data features

	Fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898	4898
mean	6.8	0.2	0.3	6.3	0.0	35.3	138.3	0.9	3.1	0.4	10.5	0.6
std	0.8	0.1	0.1	5.0	0.0	17.0	42.4	0.0	0.1	0.1	1.2	0.4
min	3.8	0.0	0.0	0.6	0.0	2.0	9.0	0.9	2.7	0.2	8.0	0.0
25%	6.3	0.2	0.2	1.7	0.0	23.0	108.0	0.9	3.0	0.4	9.5	0.0
50%	6.8	0.2	0.3	5.2	0.0	34.0	134.0	0.9	3.1	0.4	10.4	1.0
75%	7.3	0.3	0.3	9.9	0.0	46.0	167.0	0.9	3.2	0.5	11.4	1.0
max	14.2	1.1	1.6	65.8	0.3	289.0	440.0	1.0	3.8	1.0	14.2	1.0

If the features values range from a small number to very large, the importance of the features might not be correctly decided in the prediction model, see the basic statistical distribution of the data features in Table. [1]. Therefore, usually the data features are scaled using standard normal distribution, i.e. $\tilde{x} \rightarrow \frac{x - \mu}{\sigma}$ such that $\tilde{x} \in$ [-1,1], where μ and σ are the mean and standard deviation of the values x_i of a feature. We also observe the improvements in the performance of the model after features scaling, see the discussion in Section. (Results) below. One more important point to mention here is that there are 1640 bad and 3258 good wine samples in the dataset, i.e. the dataset is skewed with almost double the samples with good

wines, see Fig. (1). This fact might affect the performance of the model skewed towards the good wine due to almost double the number of such samples compared to the samples of other class.

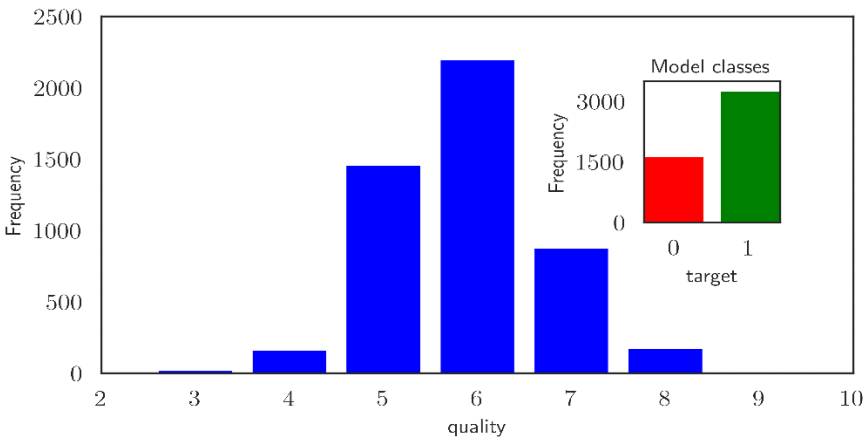

Figure 1. Bar plot of wine samples of different quality in the given dataset. The inset represents the bar plot after new target class creation based on the given quality.

Methods

k-NN algorithm is a non-parametric method used for both classification and regression in pattern recognition problems. In *k*-NN classification, the class of an object is decided by the plurality, or majority for binary classification, of the classes of *k*-nearest neighbors. It is an *instance based or lazy learning* model where the main computation is deferred until the classification and training of the model doesn't involve any heavy-lifting other than creating instances of training dataset.

In brute-force *k*-NN algorithm, the prediction on individual test sample point involves the calculation of *k*-nearest neighbors' classes of training data points based on the calculated distance between the test sample and rest of the training data points. A useful technique can be assigning the weight to the contributions of the neighbors based on their distance from the test sample point. The inverse distance weighting method gives each neighbor of weight $1/d$, where d is the distance of the neighbor from the test point, so that the nearer neighbors contribute more in decision of test point class.

If we have $\boldsymbol{n}$ testing or query points and $\boldsymbol{m}$ training or fitting points in a $\boldsymbol{f}$-dimensional space or $\boldsymbol{f}$ number of features, then $\boldsymbol{\mathcal{O}(mnf)}$ time is required for the distance computation. In addition, $\boldsymbol{\mathcal{O}(mn\ log\ m)}$ time is necessary for the sorting *k*-nearest neighbors. This implies that the total runtime complexity of the problem is $\boldsymbol{\mathcal{O}(mnf)}$ + $\boldsymbol{\mathcal{O}(mn\ log\ m)}$ for the complete computation. With $\boldsymbol{m = n}$, we can approximate the time complexity as $\boldsymbol{\mathcal{O}(fn^2)}$. For larger number of points, this algorithm quickly becomes useless in general purpose computers.

There is not any optimum value of *k*, and the best choice depends upon the data. Larger value of *k* reduces the noise in the classification with smoother but less distinct boundaries. Whereas smaller *k* leads to abrupt decision boundaries. Special case of *k =1* is called nearest-neighbor algorithm. Smaller values of *k* are better suited for the data with features that are well segregated among the data points. In such cases, increasing *k* will have negative effect but increasing *k* can be beneficial for the dataset with higher dimension and noisy features. The optimum value of *k* is usually decided by the hyper-parameter

optimization techniques, for example GridSearchCV in scikit learn. In this study, based on the analysis of performance measures on the dataset under the study, we choose *k=5*.

Inverse distance weighting: Based on the F1-score improvements with inverse distance weighting, I am using the inverse distance weighting to decide the class of the test point with different weights of its *k* neighbors with different weights based on their distance from the test point.

Results

Using the custom implemented brute-force *k*-NN classifier, we study the performance of the model using different performance metrices. The performances of the model in varying the model parameters are summarized in Table. [2]. The parameters for the best performance case are highlighted in bold text.

Table. [2]: Performance of *k*-NN model

	k	Distance	Weights	Precision	Recall	F1-Score
1	1	Euclidean	Uniform	0.855	0.846	0.850
2	**35**	**Euclidean**	**Distance**	**0.854**	**0.930**	**0.890**
3	17	Manhattan	Uniform	0.820	0.879	0.849
4	35	Manhattan	Distance	0.851	0.925	0.887
5	15	Euclidean	Uniform	0.811	0.881	0.845

From performance above, we see that the precisions are smaller than recalls i.e. the model is susceptible to higher number of false positive predictions than false negative. In current scenario, it means that the model is more vulnerable to predict a bad wine as a good wine more frequently than a bad wine good. Unfortunately, this is not a good strategy for a wine brewery company if it wants to maintain its positive relationship with costumers for long term. But from the perspective of a consumer who is sacrificing the wine quality over its price, assuming that good quality wines are more expensive than bad quality wines, the higher recall model can be more advantageous as the costumer will have higher chance of getting a good quality wine for the price of a bad quality wine.

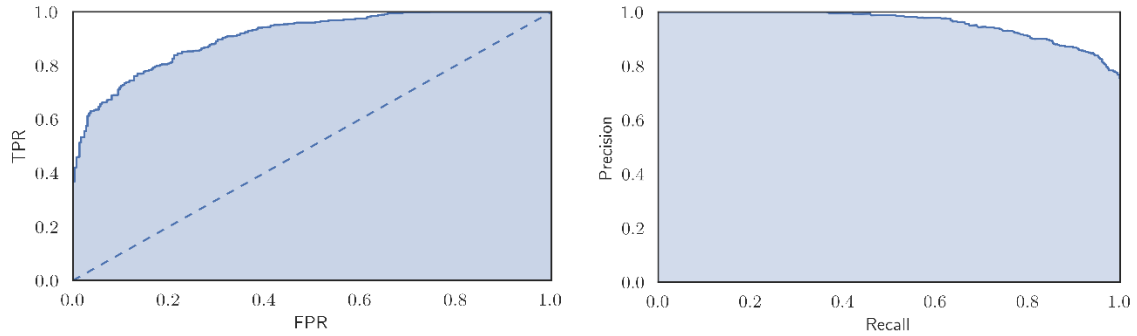

Figure 2. (Left) ROC plot with the best performance model highlighted on Table. [2]. The area of under the curve AUC is 0.90. (Right) precision-recall for the same best performance model.

Note that the model predictions changes if we carry out the data partition randomly each time as the training and testing set of sample points for each class change. Depending upon the division of entire data points

into these separate sets, we observe sometimes significant difference in the performance of the model on testing dataset. However, we can fix the repeated reproduction of same dataset using the same random state in the shuffling function, as has been done in our solution. We have presented the results with the splitting giving one of the best performances obtained from a quick experiment. But we also investigated the 95% confidence interval in the generalization errors. As shown in Fig. (3), we find generalization error in the range (0.148 – 0.192) interval with 95 % confidence interval; which means we will get the generalization errors in the predictions by the tested model fall in that range for 95% of the times we make predictions with the model.

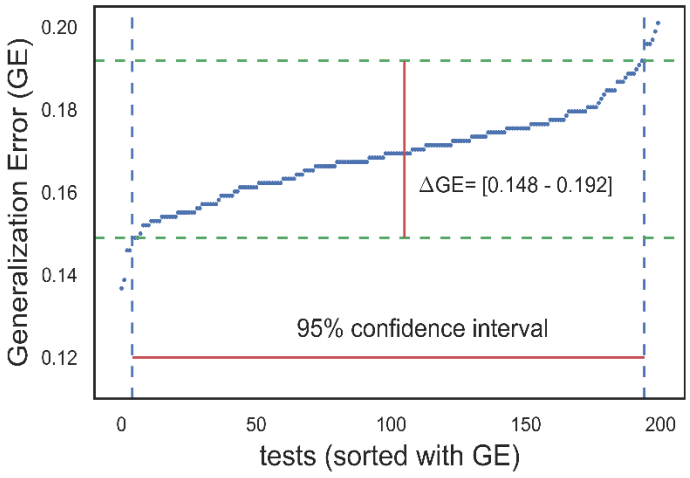

Figure 3. The 95 % confidence interval in the generalization error. The x-axis represents the tests predictions by the model using different sampled dataset sorted according to the generalization errors. ΔGE is the confidence interval for generalization error indicated by the vertical red line in the middle of the plot.

Conclusion

We implemented the brute-force *k*-NN classifier algorithm for a binary classification problem and tested the model with wine-quality dataset. We tested the correctness of our model with the pre-built version in scikit-learn and confirmed to be correctly implemented. With the simple *k*-NN classifier, we classified the wine quality into two classes: 'good' and 'bad'. After the correct features selection and hyper-parameters tuning, this simple model achieves good performance with F1-score of 0.890 in the test dataset.

REFERENCES

1. Cortez, P. (2009). *Wine Quality Dataset, https://archive.ics.uci.edu/ml/datasets/wine+quality.* UCI Machine Learning Repository.
2. Paulo Cortez, A. C. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems, 47*, 547–533.