Exploration of the KNN Algorithm and its Applications and Implementations for Big Data

Maya Hegde
Department of Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA
e-mail: mah451@pitt.edu

***Abstract*— In the early 2010's, the era of big data saw an industrial shift that allowed individuals and corporations to collect large amounts of data on activity including social media, cars, homes, accessories, and lifestyles. However, this change brought with it questions on how such large volumes of data can be handled efficiently to produce accurate results. In the computing world, this fueled a greater interest in the design of algorithms for analyzing large datasets, as well as an expansion of the applications that big data algorithms have.**

I. INTRODUCTION

This paper will further explore the K-Nearest Neighbor (KNN) algorithm and its application and implementation on real-world large datasets. The KNN algorithm is a lazy learning algorithm that is most commonly used for classification purposes in machine learning. First, this paper will discuss the basics of the KNN algorithm (Sec. II). This will lay the foundation of the first implementation of KNN to predict flower types from a dataset that contains the sepal and petal heights and widths of different irises (Sec. III). Then, I will outline the application of two different implementations of the KNN algorithm used to classify user ratings for a large Netflix dataset (Sec. IV). Finally, the different implementations of KNN will be compared (Sec. V).

II. LITERATURE REVIEW

A. Overview

To gain a better understanding of the KNN algorithm, I conducted an extensive literature review. My main goal was to learn the math behind the algorithm as well as the applications that different implementations have.

KNN is a lazy-learning algorithm, which means there is data that is used to train the model before any true test data comes in [1]. The algorithm takes in the training data, and then for each new test data that comes in, it will compare with the training data before making a classification. This is an important aspect of the algorithm because it means that it only stores the training dataset rather than going through an actual training stage, as some models and algorithms do.

KNN is considered a non-parametric, supervised-learning classifier [1]. This means that it does make assumptions about the form that the data will take and will take on any functional form from the training data [2]. This is essential because without assumptions, the algorithm can more accurately represent and classify incoming test data.

In the following two sections, I will discuss the algorithm from a high-level mathematical standpoint and then delve into the different implementations and modifications of KNN.

B. The Base Algorithm

The algorithm at the base of KNN is quite simple and consists of 6 steps [3].

1. Divide the data into two groups: test data and training data. The training data should have one or more parameters than the test data, which is what will be classification prediction.
2. Select a K value. This value will determine how many neighbors the test data point will derive.
3. Determine a distance function.
4. Choose a test point from the sample data and compute the distance between that point and all the training data points.
5. Sort the distances and take the k-nearest data samples.
6. Make a prediction for the test case based on the majority vote of the k neighbors.

As we can see, the algorithm is heavily dependent on the k value, the distance function and the prediction function. For the purposes of the algorithm analysis in this paper, the prediction function will always be a majority vote based on the k neighbors.

C. Distance Functions

There are several distance functions that can be used to calculate the differences between the test point and the training data set. For the purposes of this paper I will focus on 4: Euclidean distance, Manhattan distance, Minkowski Distance, and the Hamming distance. In Section 3 I will implement KNN with Euclidean distance, and in Section 4 I will implement KNN with both Euclidean and Manhattan distances before comparing the two in Section 5.

The Euclidean distance is the most commonly used with KNN [3]. It measures a straight line from the test point to the training point and is limited to real valued inputs.

$$Euclidean\ Distance\ =\ d(x,y)\ =\ \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \quad (1)$$

The Manhattan distance measures the absolute distance between the test and training points and gets its name from the grid-like graph it produces, which looks like city blocks [1].

$$Manhattan\ Distance\ =\ d(x,y)\ =\ (\sum_{i=1}^{n}|y_i - x_i|) \quad (2)$$

The Minkowski distance is a generalized form of the Euclidean algorithm, but allows for flexibility in adding more parameters, so I felt it was important to include it [1].

$$Minkowski\ Distance\ =\ d(x,y)\ =\ (\sum_{i=1}^{n}|y_i - x_i|)^{1/p} \quad (3)$$

The Hamming distance is typically used for non-integer vectors (such as strings or Boolean values) and it calculates the points in which the training and test points differ [1]. This distance is sometimes referred to as the overlap metric since it provides insight into the similarities of the two vectors.

$$Hamming\ Distance\ =\ D_H\ =\ (\sum_{i=1}^{k}|x_i - y_i|) \quad (4)$$

III. KNN Implementation of the Iris Dataset

A. The Iris Dataset

The iris dataset is a very popular dataset used in machine learning to test out algorithms. I used a short version of it to implement KNN so that I could visualize the algorithm and the effect of the k parameter on the algorithm. Another benefit of using the dataset is that it is smaller, so it is easier to control, validate, and analyze before using a much larger dataset.

B. Analysis and Results of the KNN Implementation of the Iris Dataset

The iris dataset contains 150 data points for training with three main classifications of irises: virginica, setosa, and versicolor. Each of these data points have 5 parameters: the sepal width, sepal height, petal width, petal height, iris classification. Then, I had three data points for testing, one that had the four parameters for a virginica iris, one that had the parameters for a setosa iris, and one that had the necessary parameters for a versicolor iris.

The algorithm was implemented using Euclidean distance calculations and a k = 5 initially. When k = 5, the algorithm was able to accurately classify all the test data. I wanted to explore more about how changing k could affect classification. Table 1, below, holds the results of the accuracy based on changing the value of k.

Table 1: Accuracy of the KNN Implementation

K value	*Correct Classification (%)*
1	0%
3	33%
5	100%
7	33%
10	0%

As we can see from Table 1, 5 seems to be the ideal k value to accurately classify data of this size. One important takeaway from this analysis is that the ideal k value is different for each data set and application.

IV. KNN Implementation of the Netflix Dataset

A. The Netflix Dataset

Between 2006 and 2009, Netflix challenged the general public to come up with an algorithm to predict user ratings for movies. The goal of this challenge was to see if there was a more efficient algorithm that could rival the one that Netflix had implemented, called Cinematch. Cinematch had a root mean square error (RMSE) of 0.9514, and Netflix promised $1 million in prize money to the team that could improve that by at least 10%.

The dataset consists of 100,480,507 ratings that 480,189 users had given to 17,770. Since this is an extremely large dataset, I wanted to use it to test out KNN using two different distance calculations: Euclidean distance and Manhattan distance. The goal of this is to determine the best distance measurement for this data and the accuracy of each fit.

B. Analysis and Results of the Euclidean and Manhattan KNN Implementations on the Netflix Dataset

The dataset was large and run time for both implementations was around three hours. Using the data given by Netflix, I was able to derive the accurate customer ratings that could be used to compare with the predicted customer ratings for each implementation. Then I compared the Euclidean and Manhattan KNN results with the correct results to derive the RMSE and accuracy of both. The final results are based on 2,135 test values. Table 2 below outlines the results.

Table 2: RMSE and Accuracy of two KNN Implementations

KNN Distance Type	*RMSE*	*Overall Accuracy*
Euclidean	1.4992582084631145	26.416861826%
Manhattan	1.4994144055993892	26.838407494%

Then, I tracked the accuracy of the two algorithms as more test data points were compared. In Figure 1, below, is a graph of those results.

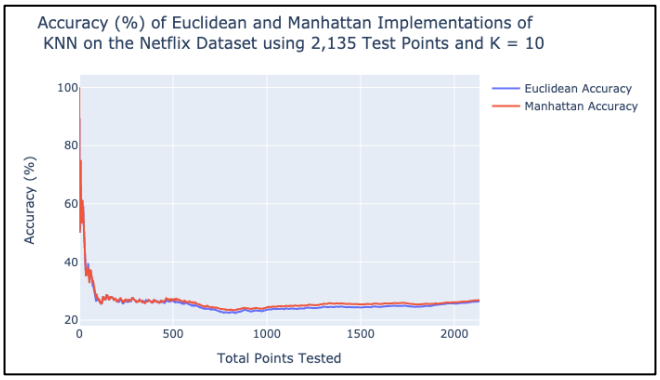

Figure 1: Accuracy of the KNN Implementations

From these results, we can conclude that the Euclidean distance provides closer predictions than the Manhattan distance within this dataset since the RMSE value for the Euclidean distance is lower than the RMSE for the Manhattan distance. However, looking solely at the accuracy of a purely correct prediction, Manhattan has a better fit than the Euclidean distance since its accuracy was consistently higher than the Euclidean over all the test data point predictions.

To mitigate the low accuracy, future steps to explore include changing the k value used and increasing the training data used. This could provide more accurate results at the expense of time.

V. Conclusion

Through the implementation and analysis of two applications of KNN algorithms, it was illustrated that KNN is a simple solution for several classification problems. However, it was also emphasized that KNN has some pitfalls. The KNN algorithm does not scale well [1]. Since it utilizes lazy learning, iterating through a lot of data takes time and memory, which can be expensive for certain applications. Additionally, it can be difficult to find the correct value of k that will allow the algorithm to fit the data the best since it depends heavily on data size and the application of the algorithm.

Another iteration of this project could provide a lot more insight into the KNN algorithm. I would focus on the effects of k on a larger dataset and also learn more about how to increase the accuracy of the algorithm without adding to time and memory costs.

VI. References

[1] "What Is the K-Nearest Neighbors Algorithm?" IBM, https://www.ibm.com/topics/knn.

[2] D. K. McIver and M. A. Friedl, "Estimating pixel-scale land cover classification confidence using nonparametric machine learning methods," in IEEE Transactions on Geoscience and Remote Sensing, vol. 39, no. 9, pp. 1959-1968, Sept. 2001, doi: 10.1109/36.951086.

[3] Zhang Z. Introduction to machine learning: k-nearest neighbors. Ann Transl Med. 2016 Jun;4(11):218. doi: 10.21037/atm.2016.03.37. PMID: 27386492; PMCID: PMC4916348.