k-Nearest Neighbors Classifier

Ali Yousefi
Machine Learning I Lab
Department of Computer Science, Bioengineering, Robotics and Systems Engineering
University of Genova
Via Balbi 5, 16126 Genova, Italy
aliyousefi98@outlook.com

***Abstract*—This experiment is a simple demonstration of implementing k-Nearest Neighbors classifier on MNIST data set. After loading the data set, k-Nearest Neighbors classifier, which is written as a MATLAB function, tries to read a random number, using the train data set.**

***Index Terms*—Classification, k-Nearest Neighbors Algorithm, MNIST data set.**

I. Introduction

In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951, and later expanded by Thomas Cover. It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.[1]

II. Approach

A. Description

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression micro-array data, for example, k-NN has been employed with correlation coefficients, such as Pearson and Spear-man, as a metric. Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighborhood components analysis.[2]

B. Details

Considering the above mentioned description, the algorithm for k-Nearest Neighbors classifier can be represented as the following three steps:

1) Load a training set and a “query” point

$$X = \{x_1, ..., x_l, ..., x_n\}, \bar{x} \tag{1}$$

2) Find the top-k nearest elements to the query point

$$\{n_1, ..., n_k\} = top - k||x_l - \bar{x}|| \tag{2}$$

3) Chose the most frequent target from top-k neighbors

$$y = mode\{t_{n_1}, ..., t_{n_k}\} \tag{3}$$

The algorithm seems to be straight forward. The most challenging step in this algorithm is finding the nearest neighbors regarding the number of features in the data set.

III. Experiments

A. Description

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training data set was taken from American Census Bureau employees, while the testing data set was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced gray-scale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.[3]

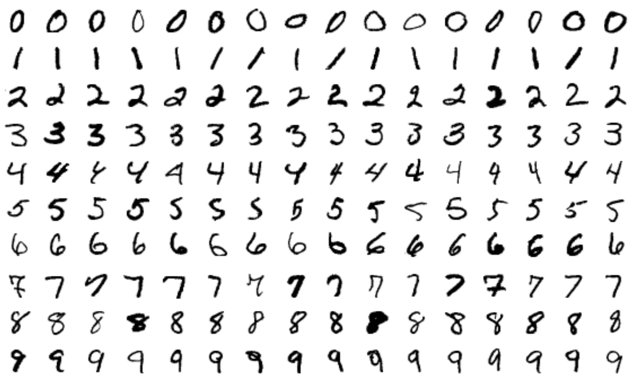

Fig. 1: MNIST Examples

The following code was used for loading the MNIST data set:

```
[X, t] = loadMNIST(1);
```

Then after choosing the size of train data set, k-NN classifier which is written as a MATLAB function, tries to guess the target value for 20 samples from the chosen train set, using different values for k:

```
train_size = 50;
for x = 1:20
    i = randi(train_size);
    for k = [1 2 3 4 5 10 15 20 30 40 50]
        y = kNN_classifier(X(1:train_size,:), t, X(i
            ,:),k);
        imshow(reshape(X(i, :), 28, 28));
        title(y);
        pause(1);
    end
end
```

The above mentioned MATLAB function for k-NN classifier is written as the following code:

```
function y = kNN_classifier(X, t, X_q, k)
    diff = diff_function(X, X_q);
    %bubble sort
    for step = 1:size(X,1)-1
        for i = 1:size(X,1)-step-1
            if diff(i) > diff(i+1)
                temp = diff(i);
                diff(i) = diff(i+1);
                diff(i+1) = temp;
                temp = t(i);
                t(i) = t(i+1);
                t(i+1) = temp;
            end
        end
    end
    t = t(1:k);
    t'
    if histc(t, mode(t))==1
        y = t(1);
    else
        y = mode(t);
    end
end
```

which utilizes another MATLAB function to find the correlation between the random chosen number and all other train data set elements. Additionally, this function uses bubble sort algorithm for sorting the found correlations, then it chooses only top-k most correlated samples and determines which is the most repeated one.

The correlation between the randomly chosen number and all other MNIST samples can be derived by finding the summation of multiplication for each two corresponding pixels, which is represented in the following MATLAB function:

```
function diff_array = diff_function(A,B)
    diff = 0;
    diff_array = zeros(size(A,1),1);
    for i = 1:size(A,1)
        for j = 1:size(A,2)
            if A(i,j)*B(j)==1
                diff = diff + 1;
            end
        end
        diff_array(i) = diff;
        diff = 0;
    end
    diff_array = size(A,2)*ones(size(A,1),1) -
        diff_array;
end
```

B. Details

The final part in this experiment was to test the written k-NN MATLAB code with the MNIST data set. Figure no. 2 shows the results which are obtained by the k-NN considering the value of k equal to 5 and with a train set size of 50 elements.

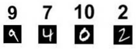

Fig. 2: Classifier Performance

obviously, k-NN classifier has a desirable performance in most cases despite its simple algorithm.

The next step was to compute k-NN classifier accuracy for different values of k. Figures no. 3 and no. 4 show the results for this experiment.

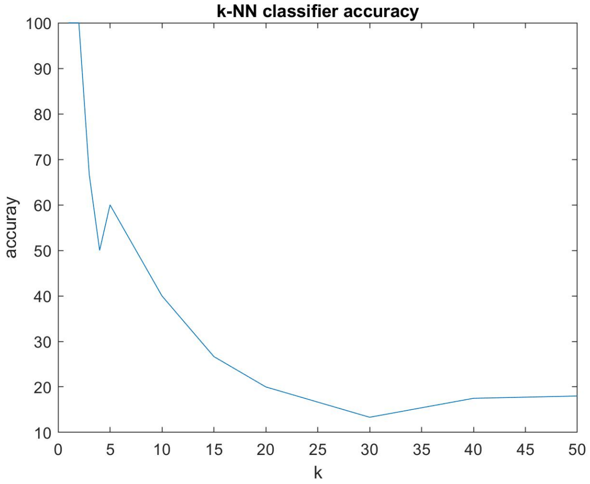

Fig. 3: k-NN Classifier Accuracy

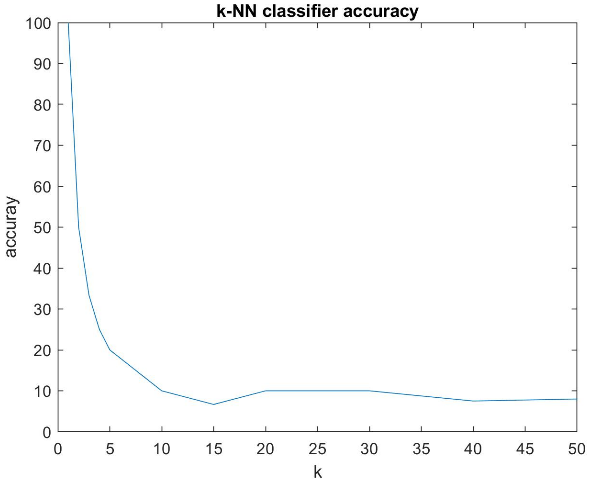

Fig. 4: k-NN Classifier Accuracy

considering the above figures, k-NN classifier accuracy drops after a while when the number of top-k neighbors increases. Additionally, for accuracy values less than 20 percent, the chosen target was often wrong.

IV. Conclusion

The k-nearest neighbors (k-NN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows. k-NN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (k) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

References

[1] Piryonesi S. Madeh; El-Diraby Tamer E. (2020-06-01). ”Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems”. Journal of Transportation Engineering, Part B: Pavements.

[2] Jaskowiak, Pablo A.; Campello, Ricardo J. G. B. ”Comparing Correlation Coefficients as Dissimilarity Measures for Cancer Classification in Gene Expression Data”. Brazilian Symposium on Bioinformatics (BSB 2011): 1–8.

[3] Cohen, Gregory; Afshar, Saeed; Tapson, Jonathan; van Schaik, André (2017-02-17). ”EMNIST: an extension of MNIST to handwritten letters”.