

K- NEAREST NEIGHBOR ALGORITHM FOR INSTANCE BASED LEARNING

CRISTINA OFELIA STANCIU

"TIBISCUS" UNIVERSITY OF TIMIȘOARA, FACULTY OF ECONOMIC SCIENCE, DALIEI STR, 1/A,
TIMIȘOARA, 300558, ROMANIA
ofelia.stanciu@gmail.com

Abstract:

Instance Based Learning (IBL) results in classifying a new instance by examining and comparing it to the rest of the instances in the dataset. An example of this type of learning is the K-Nearest Neighbor algorithm which is based on examining an average Euclidian distance of the nearest k neighbors' parameters given a certain situation.

Key words: knowledge, Instance Based Learning, algorithm, K-NN

JEL classification: D80, C38

Instance Based Learning resembles a process within which the implicit knowledge gets converted to explicit knowledge. When applying this means of learning, we start from a particular example and then generalize the explanation in order to be able to apply it in other situations as well. Adding a new rule implies a decrease in depth but also determines the increase of the number of theoretic formulas.

In practical applications, newly added rules are relevant only for certain situations, overall these rules do not make absolutely sure of their own utility. Although generally speaking, instance based learning methods have been applied in various domains and have usually proven to be effective.

An agent is able of instance based learning when it can use an example of solving a problem for solving others and this type of learning is an analytical one. The advantage of this type of learning compared to others (inductive based learning methods) is that it requires only a single learning example.

The input data the instance based learning systems benefit from are:

- *The learning example*, which is represented by a set of facts that describe an instance of the objective
- *The domain theory*, which is a set of rules describing the relations between objects and actions within a certain domain
- *The operational criteria*, displaying the form in which the definition of the generalized concept can be expressed
- *The objective*, implying a high-level description of the concept to acquire.

Instance based learning allows for the deduction of the generalization of the learning examples which is capable of describing the objective of the concept as well as of satisfying the operational criteria. This is obtained in two steps:

- Explaining – the domain theory is used in filtering learning examples with respect to the objective
- Generalization – the explanation is generalized as much as possible while continuing to describe the objective.

The K-Nearest Neighbor algorithm is one of the foundation algorithms in instance based learning [Wu 07]. This algorithm starts from associating each instance to

its corresponding point in an n -dimensional space $\mathfrak{R}^n$. The closest values of an instance are determined using the formula for Euclidian distance [Mitchel 97].

Thus, considering an instance x and $a_r(x)$ the value of the attribute r of the instance x , the following vector that describes this instance can be defined:

$$\langle a_1(x), a_2(x), \dots, a_n(x) \rangle$$

Now the distance between two instances x_i and x_j is defined like this:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^n (a_r(x_i) - a_r(x_j))^2} \quad (9)$$

In the "nearest neighbor" learning, the objective function can present discrete values as well as real ones. Given the function $f: \mathfrak{R}^n \rightarrow V$ where V is a finite set $\{v_1, v_2, \dots, v_s\}$. The KNN algorithm for approximating the objective function with discrete values is the following [Mitchel 97]:

Training algorithm:

For each training example $\langle x, f(x) \rangle$, add the example to the list training examples

Classification algorithm:

Given a query instance x_q to be classified,

Let $x_1, \dots, x_k$ denote the k instances from training examples that are nearest to x_q

Return

$$\hat{f}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

The KNN technique can be used especially for classifying data into multiple categories, but it also can be successfully applied for purposes of forecasting some objective-related attributes that have a numerical value (continuous or discrete), as a result of non-linear dependencies [Militaru 03]. This confers the possibility of performing multiple category classifications and modeling non-linear data relationship (for forecasting).

Although it is a base algorithm within instance based learning, KNN poses some disadvantages that need to be taken into account, depending on the nature of the problem. Firstly, the algorithm does not excel in computational speed when the dataset contains a large number of instances. Secondly, the algorithm is limited to supplying an estimation of the objective-attribute value without offering other information about the instance being evaluated or the dataset as a whole.

The KNN class will implement the algorithm based on finding k of the nearest neighbors of some instances in the dataset.

The first problem the algorithm implementation is going to be facing is that of transposing these instances in a normalized form, which can be represented in a two-dimensional space as points with their corresponding coordinates. The problem lies in resolving these coordinates in order to be able to apply the geometrical formulas that represent the steps of this algorithm.

For this purpose, the KNN class will first define two helper methods, *CalculateMean* and *CalculateStandardDeviation* with their implementation:

```

private double CalculateMean(Instances set,
                             string attributeName)
{
    double result = 0;
    foreach (Instance data in set)
    {
        result += data[attributeName].NumericValue;
    }
    return result / set.Count;
}

```

and respectively:

```

private double CalculateStandardDeviation(Instances set,
                                          string attributeName)
{
    double result = 0;
    double mean = CalculateMean(set, attributeName);
    foreach (Instance data in set)
    {
        result += Math.Pow(
            data[attributeName].NumericValue - mean, 2);
    }
    return Math.Sqrt(result / set.Count);
}

```

The first method computes the arithmetic average of the numerical projection of an attribute value, projection that is identified by the *NumericValue* property. The value of this property is supplied by the following mathematical expression:

$$\bar{A} = \frac{\sum_{v \in A} v}{|S|} \quad (10)$$

where v will be iterating through all the possible values of attribute A and S is the dataset.

The second helper method computes the standard deviation of an attribute using the following formula:

$$\sigma_A = \sqrt{\frac{\sum_{v \in A} (v - \bar{A})^2}{|S|}} \quad (11)$$

These values are needed in order to be able to scale the dataset in such manner that the global arithmetic average of the entire dataset will be 0 and the standard deviation will be 1. This scaling is performed by replacing the numeric value of each attribute with a new value obtained using the following calculation:

$$v' = \frac{v - \bar{A}}{\sigma_A} \quad (12)$$

where v represents the value of the attribute A , which will be replaced with its new value v' .

The implementation will have to create a method that will generate a matrix representation of the initial dataset:

```
scaledValues = new Dictionary<AttributeDefinition,
                        Dictionary<Instance, double>>(
                        relation.Definitions.Count);
double mean, stdDev;
foreach (AttributeDefinition def in relation.Definitions)
{
    scaledValues[def] = new Dictionary<Instance, double>(
                        relation.Instances.Count);
    mean = CalculateMean(relation.Instances, def.Name);
    stdDev = CalculateStandardDeviation(relation.Instances, def.Name);
    foreach (Instance data in relation.Instances)
    {
        scaledValues[def][data] = (data[def.Name].NumericValue-mean) /
stdDev;
    }
}
```

This matrix structure will be represented through the *Dictionary* classes, thus offering the possibility to access a value based on its attribute and the instance is can be found in.

The *KNN* algorithm relies on evaluating distances between two points in the dataset plan; hence the need of a method that can compute this distance. Given that there is already a matrix containing the scaled values of the dataset and these can be easily accessed based on their initial source, the *CalculateDistance* method will be defined with two parameters of type *Instance*, which will be responsible for performing all the required projections for the instances; an invocation of this method could be interpreted as "*evaluate the distance between instances x and y*". The method prototype and implementation will look like this:

```
private double CalculateDistance(Instance data1, Instance data2)
{
    double result = 0;
    foreach (AttributeDefinition def in relation.Definitions)
    {
        if (def.Conclusive)
            continue;

        result += Math.Pow(
            scaledValues[def][data1] - scaledValues[def][data2], 2);
    }
    return Math.Sqrt(result);
}
```

The method uses the Euclidian procedure for computing the distances; the distance between two instances in the dataset identified by the (scaled) values of their attributes is defined by the Euclidian geometry through the following formula:

$$d(a,b)=\sqrt{\sum_A (a'_A-b'_A)^2} \quad (13)$$

where:

- a and b are instances within the dataset
- A will iterate though each attribute values in the dataset
- x_A represents the value of attribute A for instance x ; x' is the scaled value of that attribute

At this point the algorithm has all the input data and required methods at its disposal and awaits execution. The following piece of code can find k of the nearest neighbors of any given instance not belonging to the initial dataset (k can be passed as a parameter):

```
Instance[] nearest = new Instance[neighborCount];
Instances set = new Instances(relation.Instances);
for (int k = 0; k < neighborCount; k++)
{
    CalculateShortestDistance(set, testData, out nearest[k]);
    set.Remove(nearest[k]);
}
```

The *CalculateShortestDistance* method will call the *CalculateDistance* method, successively comparing distances between the supplied instance and all instances within a given data subset and picking out the one where the distance hits a minimum value. Once it finds the closest instance, it will remove it from a temporary set, thus preventing it to be found in subsequent runs.

When this piece of code finishes execution, the result will be represented by a list of k of the closest instances in the *nearest* variable. At this point, these neighbors can be analyzed with respect to the objective-attribute within each instance in order to discover the most frequent conclusion. This analysis can be achieved through a simple classic algorithm that discovers the maximum value in an array, but prior to that, the identical conclusions for the k nearest neighbors will need to be counted and retained.

BIBLIOGRAPHY:

1. Cojocariu, A., A. Munteanu, Sofran (Stanciu), Cristina-Ofelia, *Machine Learning techniques for enhancing decision support systems within organizations*, 29th International Convention MIPRO, Conference on Business Intelligence Systems (BIS), Opatija, Croatia, May 22 -26, 2006
2. HyounghDo, K., *An XML-based modeling language for the open interchange of decision models*, Decision Support Systems, Volume 31, Issue 4, Pages 429-441, October 2001
3. Militaru, V., *Studiu comparat asupra tehnicilor de data mining utilizate în rezolvarea problemelor de regresie și clasificare*, Revista Informatica Economica, nr. 3(27), 2003
4. Mitchell, T., *Machine Learning*, McGraw – Hill Publishing, 1997
5. Stanciu, Cristina – Ofelia, A. Cojocariu, *XML Technologies for Improving Data Management for Decision Algorithms*, 20th DAAAM International Conference, Vienna, 2009
6. Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z., Steinbach, M., Hand · D. J., Steinberg, D., *Top 10 algorithms in data mining*, Knowl Inf Syst 14:1–37, Springer Publishing, 2008
7. *** <http://kdd.ics.uci.edu/databases/tic/tic.html>