K-Nearest Neighbors (KNN)

Mads Møller

LinkedIn: https://www.linkedin.com/in/madsmoeller1/

Mail: mads.moeller@outlook.com

This paper is the fifth in the series about machine learning algorithms. The Nearest Neighbor algorithm is used for *classification* problems but can also be used for regression. Nearest Neighbor is also a supervised machine learning algorithm, meaning that our data need to be labelled. The Nearest Neighbor algorithm is one of the most popular algorithms within classification. It is moreover a multi-class classification algorithm.

1 Dissimilarities and Distances

The intuition behind behind the KNN algorithm is reflected in its name. We measure how much each observation are similar to other observations. In KNN, similar observations are assigned to the same class. **Similarity** is the keyword when we talk about KNN. There are a lot of different methods of how you can measure similarity between observations, we will dive into some of these methods in a moment. The complement of *similarity* is *dissimilarity* and when we talk about KNN it is actually the dissimilarity we are interested in. We define a dissimilarity as a function δ that satisfy:

- $\delta_{ij} \geq 0$
- $\delta_{ii} = 0$
- $\delta_{ij} = \delta_{ji}$

Where i and j are different observations.
We can from the dissimilarity function see that one observation is equal to (100% similar or 0% dissimilar) another observation if $\delta_{ij} = 0$. Some well-known distances for continous variables are *Euclidean* distance, *Manhattan* distance and *Maximum* distance. All of these three distances belongs to a family of distances called l_p-distances.

1.1 Distances for continous variables

The first distance we will take a look at is the **Euclidean**:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ik} - x_{jk})^2} \tag{1}$$

In a 2D or 3D setting the euclidean distance can be seen as the length of the straight line that connects two observations in our feature space.

The **Manhattan** distance is very similar with the *Euclidean distance*. We can calculate the Manhattan distance in the following way:

$$d(\mathbf{x}_i, \mathbf{x}_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ik} - x_{jk}| \tag{2}$$

The last distance measure we will see is the **Maximum** distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \max\{|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \ldots, |x_{ik} - x_{jk}|\} \tag{3}$$

You can think of the Maximum distance as the worst case, or the longest distance for a variable between two observations. As mentioned earlier all of these distances belongs to a class of distances called l_p-distance. The general formulation of l_p-distance is:

$$d(\mathbf{x}_i, \mathbf{x}_j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \ldots + |x_{ik} - x_{jk}|^p)^{1/p} \tag{4}$$

From the general formulation we can see that for $p = 2$ we have the *Euclidean*, for $p = 1$ we have the *Manhattan* and for $p = \infty$ we have the *Maximum*.

1.2 distances for categorical variables

Until now we have only been looking at how we can measure distance between continuous variables. However, in many cases a dataset also contains categorical variables. We also need to be able to measure how "far" an observation is from another observation if we have a categorical variable. There are three popular measurements of distances for categorical variables. These are called *Hammig*, *Simple Matching* and *Jaccard*. The **Hamming** distance is based on counting:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \text{cardinality}\{s : x_{is} \neq x_{js}\} \tag{5}$$

It is counting the number of differences in categorical variables between two observations. So, imagine a dataset with three categorical variables. If two observations only has the same category in one of the three variables then the Hamming distance would be 2. Because there are two categories where the observations are different. The other two measurements are simple as well but we will not go into these in this paper.

When you have a dataset with both continuous and categorical variables normally the algorithm within the program you use will do a mixture of distances in order to handle both data types.

2 The KNN Algorithm

The KNN algorithm is based on **similarity** arguments. The KNN algorithm does not deliver a model per se, but it is an algorithm to classify new observations in an already existing dataset. Similarity defines the "neighbors" in KNN. The neighbors are the most similar observations to a new observation. The **K** in KNN is a constant (hyperparameter) that defines how many neighbors the algorithm should find. The "problem" with KNN is that it is a computational expensive algorithm.

2.1 Class Assignment

For new observations, the predicted class is defined by the K closest (most similar) neighbors. How KNN is classifying a new observation is by looking at the majority class in the K-nearest neighbors. The new observation will be assigned to the same class as the majority of its neighbors. Given a new individual $\mathbf{x}_{new}$ the KNN algorithm is calculating $\delta_{new,i}$ for all i in the training dataset. Afterwards the dissimilarities are being rearranged in increasing order, e.g.

$$\delta_{new,(1)} \leq \delta_{new,(2)} \leq \cdots \leq \delta_{new,(K)} \leq \delta_{new,(K+1)} \leq \cdots \leq \delta_{new,(n)}$$

The K first observations are the so-called nearest neighbors. $\hat{y}_{new}$ is the majority class among the K nearest neighbors.

2.2 Hyperparameter Tuning

As in almost all machine learning models there are hyperparameters we can tune to increase our "model"-performance. In case of KNN we do not really have a model, but we can tune some hyperparameters to make our classification performance better. The number of neighbors, K, is the most obvious hyperparameter we can tune. You can also tune the p in l_p-distance measurement.