K-Nearest Neighbor (KNN) Algorithm

Contents:

What is KNN algorithm?

• **KNN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure**.

• Let's assume we have several groups of labeled samples. The items present in the groups are homogeneous in nature. Now, suppose we have

an unlabeled example which needs to be classified into one of the several labeled groups. How do we do that? Unhesitatingly, using kNN Algorithm

• **In a single sentence, nearest neighbor classifiers are defined by their characteristic of classifying unlabeled examples by assigning them the class of similar labeled examples.**

• The k-NN algorithm gets its name from the fact that it uses information about an example's k-nearest neighbors to classify unlabeled examples

• The letter k is a variable term implying that any number of nearest neighbors could be used. After choosing k, the algorithm requires a training dataset made up of examples that have been classified into several categories, as labeled by a nominal variable.

When do we use KNN?

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time

3. Predictive Power

KNN can be used for regression and classification problems:

KNN for Regression

• When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

KNN for Classification

• When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

How does the KNN algorithm work?

Let’s take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):

You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The “K” in KNN algorithm is the nearest neighbors we wish to take vote from. Let’s say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:

The three closest points to BS are all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm.

Example of kNN Algorithm

Let’s consider 10 ’drinking items’ which are rated on two parameters on a scale of 1 to 10. The two parameters are “sweetness” and “fizziness”.. The ratings of few items look somewhat as:

“Sweetness” determines the perception of the sugar content in the items. “Fizziness” ascertains the presence of bubbles in the drink due to the carbon dioxide content in the drink.

From the above figure, it is clear we have bucketed the 10 items into 4 groups namely, ’COLD DRINKS’, ‘ENERGY DRINKS’, ‘HEALTH DRINKS’ and ‘HARD DRINKS’. The question here is, to which group would ‘Maaza’ fall into? This will be determined by calculating distance.

Calculating Distance

Now, calculating distance between ‘Maaza’ and its nearest neighbors (‘ACTIV’, ‘Vodka’, ‘Pepsi’ and ‘Monster’) requires the usage of a distance formula, the most popular being Euclidean distance formula i.e. the shortest distance between the 2 points which may be obtained using a ruler.

Using the coordinates of Maaza (8,2) and Vodka (2,1), the distance between

‘Maaza’ and ‘Vodka’ can be calculated as:

dist(Maaza,Vodka) = 6.08

Using Euclidean distance, we can calculate the distance of Maaza from each of its nearest neighbors. Distance between Maaza and ACTIV being the least, it may be inferred that Maaza is same as ACTIV in nature which in turn belongs to the group of drinks (Health Drinks).

If k=1, the algorithm considers the nearest neighbor to Maaza i.e, ACTIV; **if k=3, the algorithm considers '3' nearest neighbors to Maaza to compare the distances (ACTIV, Vodka, Monster) – ACTIV stands the nearest to Maaza.**

Choosing an appropriate k:

• The decision of how many neighbors to use for k-NN determines how well the model will generalize to future data.

• The balance between overfitting and underfitting the training data is a problem known as **bias-variance tradeoff**.

• Choosing a large *k* reduces the impact or variance caused by noisy data, but can bias the learner so that it runs the risk of ignoring small, but important patterns.

• Suppose we took the extreme stance of setting a very large *k*, as large as the total number of observations in the training data.

• **With every training instance represented in the final vote, the most common class always has a majority of the voters**. The model would consequently always predict the majority class, regardless of the nearest neighbors.

• **On the opposite extreme, using a single nearest neighbor allows the noisy data or outliers to unduly influence the classification of examples. For example, suppose some of the training examples were accidentally mislabeled. Any unlabeled example that happens to be nearest to the**

incorrectly labeled neighbor will be predicted to have the incorrect class, even if nine other nearest neighbors would have voted differently.

• **One common practice is to begin with *k* equal to the square root of the number of training examples**.

• **A less common, but interesting solution to this problem is to choose a larger *k*, but apply a weighted voting process in which the vote of the closer neighbors is considered more authoritative than the vote of the far away neighbors.** Many k-NN implementations offer this option

• If K=1, select the nearest neighbor

• If K>1, – For classification select the most frequent neighbor. –

• For regression calculate the average of K neighbors.

Why is KNN called as a Lazy learning algorithm?

• **KNN is a non *parametric lazy learning* algorithm.**

• **Non parametric, it means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as in the real world , most of the practical data does not obey the typical theoretical assumptions made (e.g. Gaussian mixtures, linearly separable etc).**

• As instance-based learners do not build a model, the method is said to be in a class of non-parametric learning methods—**no parameters are learned about the data. Without generating theories about the underlying data.**

• It is also **a lazy algorithm**. What this means is that it does not use the training data points to do any *generalization*.

• K-NN is a lazy learner because it doesn't learn **a discriminative function from the training data but "memorizes" the training dataset instead.**

• No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.

• Lazy learning is also known as **instance-based learning or rote learning.**

• For example, the logistic regression algorithm learns its model weights (parameters) during training time. In contrast, there is no training time in K-NN. Although this may sound very convenient, this property doesn't come without a cost: The "prediction" step in K-NN is relatively expensive! Each time we want to make a prediction, K-NN is searching for the nearest neighbor(s) in the entire training set! (Note that there are certain tricks such as BallTrees and KDtrees to speed this up a bit.)

• **"Lazy learners" do not require building a model with a training set before actual use. A lazy learner like k-nearest neighbors uses the training set directly to classify an input when an input is given**. When using a k-nearest neighbor algorithm on an input with d attributes the input is classified by taking a majority vote of the k (where k is some user specified constant) closest training records across all d attributes. "Closest" as used means the distance an attribute is away from the same attribute of the training set, using some specified similarity metric.

For each data point, the algorithm finds the k closest observations, and then classifies the data point to the majority. Usually, the k closest observations are defined as the ones with the smallest Euclidean distance to the data point under consideration. For example, if k = 3, and the three nearest observations to a specific data point belong to the classes A, B, and A respectively, the algorithm will classify the data point into class A.

Then, for each unlabeled record in the test dataset, k-NN identifies k records in the training data that are the "nearest" in similarity. The unlabeled test instance is assigned the class of the majority of the k nearest neighbors.

KNN DIFFERENT NAMES:

K-Nearest Neighbors

Memory-Based Reasoning

Example-Based Reasoning

- Instance-Based Learning
- Case-Based Reasoning

- Lazy Learning

KNN Application:-Dout

Strength and Weakness of KNN:

Case Study: Detecting Prostate Cancer

Machine learning finds extensive usage in the pharmaceutical industry especially in detection of oncogenic (cancer cells) growth. R finds application in machine learning to build models to predict the abnormal growth of cells thereby helping in detection of cancer and benefiting the health system.

Let's see the process of building this model using kNN algorithm in R Programming.

Step 1- Data collection

The data set consists of 100 observations and 10 variables (out of which 8 numeric variables and one categorical variable and is ID) which are as follows:
1. Radius 2. Texture 3. Perimeter 4. Area 5. Smoothness 6. Compactness 7. Symmetry 8. Fractal dimension

Step 2- Preparing and exploring the data :

We find that the data is structured with 10 variables and 100 observations. If we observe the data set, the first variable 'id' is unique in nature and can be removed as it does not provide useful information.

The data set contains patients who have been diagnosed with either Malignant (M) or Benign (B) cancer

(The variable diagnosis result is our target variable i.e. this variable will determine the results of the diagnosis based on the 8 numeric variables)

Normalizing numeric data

This feature is of paramount importance since the scale used for the values for each variable might be different. The best practice is to normalize the data and transform all the values to a common scale.

The first variable in our data set (after removal of id) is ‘diagnosis_result’ which is not numeric in nature. So, we start from the 2nd variable. The function lapply() applies normalize() to each feature in the data frame. The final result is stored to cancer_data_n data frame using as.data.frame() function

Creating training and test data set

The kNN algorithm is applied to the training data set and the results are verified on the test data set. We would divide the data set into 2 portions in the ratio of 65: 35 (assumed) for the training and test data set respectively.

A blank value in each of the above statements indicates that all rows and columns should be included.

Our target variable is ‘diagnosis_result’ which we have not included in our training and test data sets.

Step 3 – Training a model on data

The **knn ()** function needs to be used to train a model for which we need to install a package ‘class’

The **knn()** function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number.

The value for k is generally chosen as the square root of the number of observations.

knn() returns a factor value of predicted labels for each of the examples in the test data set which is then assigned to the data frame cancer_test_pred

Step 4 – Evaluate the model performance

We have built the model but we also need to check the accuracy of the predicted values in prc_test_pred as to whether they match up with the known values in prc_test_labels. To ensure this, we need to use the **CrossTable()** function available in the package '**gmodels'.**

The test data consisted of 35 observations. Out of which 5 cases have been accurately predicted (TN->True Negatives) as Benign (B) in nature which constitutes 14.3%. Also, 16 out of 35 observations were accurately predicted (TP- > True Positives) as Malignant (M) in nature which constitutes 45.7%

The cells falling on the other diagonal contain counts of examples where the k-NN approach disagreed with the true label. The one example in the lower-left cell are **false negative** results; in this case, the predicted value was benign, but the tumor was actually malignant. Errors in this direction could be extremely costly as they might lead a patient to believe that she is cancer-free, but in reality, the disease may continue to spread. The top-right cell would contain the **false positive** results, if there were any. These values occur when the model classifies a mass as malignant, but in reality, it was benign. Although such errors are less dangerous than a false negative result, they should also be avoided as they could lead to additional financial burden on the healthcare system or additional stress for the patient as additional tests or treatment may have to be provided.

There were no cases of False Negatives (FN) meaning no cases were recorded which actually are malignant in nature but got predicted as benign. The FN's if any poses a potential threat for the same reason and the main focus to increase the accuracy of the model is to reduce FN's.

There were 14 cases of False Positives (FP) meaning 14 cases were actually benign in nature but got predicted as malignant.

The total accuracy of the model is 60 %((TN+TP)/35) which shows that there may be chances to improve the model performance

Step 5 – Improve the performance of the model

This can be taken into account by repeating the steps 3 and 4 and by changing the k-value. Generally, it is the square root of the observations and in this case we took k=10 which is a perfect square root of 100.The k-value may be fluctuated in and around the value of 10 to check the increased accuracy of the model. Do try it out with values of your choice to increase the accuracy! Also remember, to keep the value of FN's as low as possible.

As the k-means algorithm is highly sensitive to the starting position of the cluster centers, this means that random chance may have a substantial impact on the final set of clusters.