

Machine Learning: KNN Algorithm a Review

Satyanarayana Medicherla

Consultant, Canon U.S.A. Inc, 1, Canon Park, New York - 11747, email- satyams@hotmail.com

Abstract

K-Nearest Neighbors is one of the simplest Supervised Machine Learning algorithms mostly used for classification problems. KNN involves finding similar items. The distance or similarity is measured by comparing the feature vectors. While comparing the feature vectors, a distance measure like Euclidian distance or Cosine similarity is used. K-Nearest Neighbors is a non-parametric method where we do not assume any distribution of the underlying data for which we find the parameters [1]. In this article, we define distance/similarity measure and then study some test data and discuss this algorithm in detail. Some applications of KNN algorithm are Credit Rating, Loan Application Processing and Voter Classification

Introduction

K-Nearest Neighbors (KNN) is a non-parametric technique used mostly in classification problems. KNN is a lazy learner because it does not create a model and use the model for prediction [2]. It is a brute-force method because this has to calculate the distance from all the training examples before deciding the K nearest neighbors. When we want to classify a new example, it computes some distance metric to find out the training examples those closely resemble our query example. The new query example will be classified as the majority class out of the K nearest training examples. Here K is a positive integer. We will discuss the ways to determine the optimal K for any given data set. We will discuss different distance metrics used in KNN.

KNN Algorithm

KNN is a supervised form of learning i.e. the data labels are supplied with the training data set. Let us discuss how this algorithm works towards finding similar items from the set of items provided in the form of feature vectors. We are given a set of training examples with a set of feature vectors X and the outcome label Y. Our goal is to find a mapping function F that maps the feature vector X to the label vector Y [2].

$$F(x_i): X \rightarrow Y$$

The feature vector is of the form

$$x_1 = (x_{11}, x_{12}, x_{13}, \dots, x_{1n})$$

$$x_2 = (x_{21}, x_{22}, x_{23}, \dots, x_{2n})$$

$$x_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{in})$$

$$x_j = (x_{j1}, x_{j2}, x_{j3}, \dots, x_{jn})$$

$$x_m = (x_{m1}, x_{m2}, x_{m3}, \dots, x_{mn})$$

Now we need to compute the distances of our query example from all the training examples. Find the K training examples with the least distance from the example that we are classifying. The majority class of those K training examples is output as the estimated class. The distance measure may not always be the Euclidean distance and sometimes the Cosine similarity is more meaningful depending upon the situation. We will see how to deal with non-numeric features while calculating the distance or similarity.

Let us say we want to find the distance between the examples x_i and x_j

We can find the distance between x_i and x_j using any of the following:

The Euclidean distance between the two examples x_i and x_j is

$$D(x_i, x_j) = \sqrt{\sum_{k=1}^n (x_{ik})^2 - x_{jk})^2}$$

or Cosine Similarity between x_i and x_j is

$$\text{Cosine Similarity } (x_i, x_j) = \frac{\sum_{i=1}^n x_{ik} * x_{jk}}{\sqrt{(\sum_{i=1}^n (x_{ik})^2) * (\sum_{i=1}^n (x_{jk})^2)}}$$

or any other distance metric discussed later in this document.

Pros and Cons of KNN

KNN is one of the simplest machine learning algorithms and very easy to comprehend and implement [17]. It does not assume any distribution for the underlying data. There is no training step involved. KNN is memory intensive and it has to retain all the data in during prediction phase also since there is no training step. We need to scale the features before applying KNN to make the features to have the same scale. Since the neighbors are determined based on distance, KNN is very sensitive to outliers.

Distance Calculation

KNN-algorithm is driven by the distance calculation between our query example from all of our training examples. The distance function is used for deciding the nearness in K-Nearest Neighbors algorithm. There are different distance metrics that we can use based on the context. Let us discuss different distance functions and their usage.

Euclidean Distance

The simplest one is the Euclidean Distance, which is the basic straight-line distance between two points in Euclidean Space [15]. When the features are continuous numeric values, we can use the Euclidean Distance Metric. It is computed as follows in two-dimensional space. The Euclidean distance between (x_1, y_1) and (x_2, y_2)

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Treats all dimensions equally [10] [11] and it is sensitive to extreme differences in a single attribute. To reduce the attribute sensitivity, we need to standardize the attributes using one of the few methods discussed later in this document.

Manhattan Distance

Manhattan Distance is defined as the sum of the differences between the corresponding components of two vectors. For example, if you have two different vectors (x_1, x_2) and (y_1, y_2) , the Manhattan distance is

The Manhattan distance between (x_1, y_1) and (x_2, y_2)

$$|x_1 - x_2| + |y_1 - y_2|$$

ChebyChev's Distance

ChebyChev's distance between two vectors is the greatest of their differences along any coordinate dimension [9]. It is named after Pafnuty Chebyshev. Let us say the vectors are (x_1, y_1) and (x_2, y_2) then the ChebyChev's distance is given by

The Chebychev's distance between (x_1, y_1) and (x_2, y_2)

$$\max(|x_1 - x_2|, |y_1 - y_2|)$$

Cosine Similarity

In some cases, the Euclidean distance is not very meaningful. Let us say we are classifying a document using the KNN- algorithm. The documents are represented in vector form with word counts as the components of the vectors [8]. When plotted in a multi-dimensional space, the magnitude of the vectors cannot capture the similarity but the cosine similarity can capture the similarity better.

$$\text{Cosine similarity} = \frac{A \cdot B}{\|A\| \|B\|}$$

where A and B feature vectors, A.B is the dot product of the vectors and $\|A\|$ and $\|B\|$ are norms or magnitudes of the vectors.

$$A = (a_1, a_2, a_3, a_4, \dots, a_n) \quad B = (b_1, b_2, b_3, b_4, \dots, b_n)$$

$$\text{Cosine Similarity (A,B)} = \frac{\sum_{i=1}^n A_i \cdot B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

Hamming Distance

We will use the Hamming Distance for comparing two binary strings of digits. It is the number of bits that the two binary strings differ [10]. This distance is used in case of encoded categorical variables. In case of encoded categorical variables with 3 different possible values, We will add 3 different new binary columns to the data set. Depending upon the value of the original categorical variable, the appropriate column will take a value of 1 and the rest would be made zeroes.

The Hamming distance between two strings, a and b is denoted as $d(a,b)$.

Let us take a simple example

$$a = 1011 \ 1100 \ \text{and} \ b = 0101 \ 0101$$

Let us find the difference using not-equal-to or XOR, which outputs a 1 only when both the binary digits

being compared are different.

a XOR b = 1110 1001

There are five 1's in the output indicating that the two binary string differ in 5 binary digits. So the Hamming Distance is 5.

Standardization of feature vector

During classification, we compute the distance using the features vectors. If the range of some of the features is much more when compared to that of the other features [3], the distance computation is mostly influenced by the feature with maximum range. The classification is solely driven by that feature. In order to reduce this effect, we need to standardize the features.

```
m_data <-
matrix(c(1,1,1,1,1,100,"y1",2,2,2,2,2,104,"y2",1,1,1,1,
1,105,"y1"),ncol=7,byrow=TRUE)
colnames(m_data) <-
c("f1","f2","f2","f4","f5","f6","y")
rownames(m_data) <-c("1","2","3")
t_data <-as.table(m_data)
t_data

## f1 f2 f2 f4 f5 f6 y
## 1 1 1 1 1 1 100 y1
## 2 2 2 2 2 2 104 y2
## 3 1 1 1 1 1 105 y1
```

Let us say we are calculating the distance between the sample 3 and the other samples 1 and 2

Distance between 3 and 1

$$\sqrt{((1-1)^2 + (1-1)^2 + (1-1)^2 + (1-1)^2 + (1-1)^2 + (105-100)^2)}$$

$$\sqrt{(25)}$$

Distance between 3 and 2

$$\sqrt{((1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (105-104)^2)}$$

$$\sqrt{(6)}$$

So the sample 3 looks to be close to 2, though it is closer to sample-1 because it agrees on 5 of the 6 attributes with sample-1. This is because of the very high range of the feature f5. That is why feature scaling is very important.

Using Z-score

We will subtract the mean of the column from each value and divide it by the standard deviation.

$$x_i = \frac{(x_i - \text{mean}(x_i))}{sd(x_i)}$$

Mean Normalization

Similar to above but we will divide by the range of the values.

$$x_i = \frac{(x_i - \text{mean}(x_i))}{(\text{max}(x_i) - \text{min}(x_i))}$$

Min-Max Normalization

We will subtract the minimum value from each values and divide by the range of the values.

$$x_i = \frac{(x_i - \text{min}(x_i))}{(\text{max}(x_i) - \text{min}(x_i))}$$

Unit-Vector Normalization

We will divide each value by the norm or magnitude of the vector.

$$x_i = \frac{x_i}{||x_i||}$$

Categorical Variables

Distance computation is the core of the KNN-algorithm. Categorical or Qualitative Variables are non-numeric variables i.e. they take on values that are names or labels [11]. We cannot compute distance for categorical attributes. For example the names of cities or colors of flowers. Since these do not have numeric values and these would pose problems distance calculation between two attribute vectors. Now let us discuss some encoding methods that are employed for computing the distance calculation in case of categorical variables.

Some modelling techniques like Decision Tree Classification can work directly on Categorical variables but techniques like KNN cannot work with categorical variables directly because the distance cannot be computed for them.

Target Based Encoding

In case of target based encoding, each categorical variable is replaced by its corresponding probability

[16]

```
m_data <-
matrix(c('Up',1,0.50,'Up',1,0.50,'Up',0,0.50,'Down',0,0
.66,'Down',1,0.66,'Up',0,0.50,'Up',0,0.50,'Down',1,0.6
6),ncol=3,byrow=TRUE)
colnames(m_data) <-c("Col","target","col_enc")
rownames(m_data) <-
c("1","2","3","4","5","6","7","8")
t_data <-as.table(m_data)
t_data

## Col target col_enc
## 1 Up 1 0.5
## 2 Up 1 0.5
## 3 Up 0 0.5
## 4 Down 0 0.66
## 5 Down 1 0.66
## 6 Up 0 0.5
## 7 Up 0 0.5
## 8 Down 1 0.66

m_data <-
matrix(c('Up','2','2','0.50','Down','1','2','0.66'),ncol=4,b
yrow=TRUE)
colnames(m_data) <-c("Col","0","1","Prob")
rownames(m_data) <-c("1","2")
t_data <-as.table(m_data)
t_data

## Col 0 1 Prob
## 1 Up 2 2 0.50
## 2 Down 1 2 0.66
```

One-hot Encoding for Categorical Variables

Categorical variables take on the values that are names or labels [12]. Let us say the column is color, which can take on 3 different values like Red, Blue and Green. We will create a new column for each of the different values of the given column.

```
m_data <-
matrix(c("Red",1,0,0,"Blue",0,1,0,"Green",0,0,1,"Red
",1,0,0,"Blue",0,1,0,"Green",0,0,1),ncol=4,byrow=TR
UE)
colnames(m_data) <-
c("Color","Color_Red","Color_Blue","Color_Green")
rownames(m_data) <-c("1","2","3","4","5","6")
t_data <-as.table(m_data)
t_data

## Color Color_Red Color_Blue Color_Green
## 1 Red 1 0 0
## 2 Blue 0 1 0
```

```
## 3 Green 0 0 1
## 4 Red 1 0 0
## 5 Blue 0 1 0
## 6 Green 0 0 1
```

The distance between any two rows in the above table is 2, if the values are different or 0 if they are same. If we have many different values for the variable being encoded, we will need to create as many new variables.

KNN-algorithm Examples

We will apply KNN for different data sets to study this algorithm further. We will use a data set with all numerical features in the first example and then we will use a data set with numerical and categorical data set. We will see how we can make use of different techniques discussed so far.

All Numeric or continuous data

We will use the iris data set shipped with R for discussing KNN-algorithm. Here is a brief description of this data that is provided by R.

Description provided by R

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Usage

iris

iris is a data frame with 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
 - Iris Setosa
 - Iris Versicolour
 - Iris Virginica

Let us

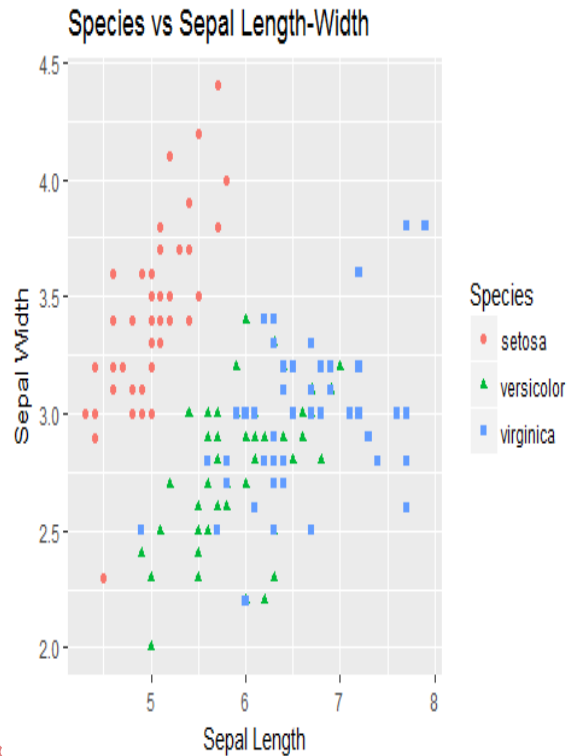
```
data("iris")
```

```
str(iris)

## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4
4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4
2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5
1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2
0.2 0.1 ...
## $ Species : Factor w/ 3 levels
"setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 ...
```

```
library(ggplot2)
```

```
par(mfrow=c(1,2))
scatter1 <-ggplot(data=iris, aes(x = Petal.Length, y =
Petal.Width)) +
geom_point(aes(color=Species, shape=Species)) +
xlab("Petal Length") +ylab("Petal Width") +
ggtitle("Species vs Petal Length-Width")
scatter1
```



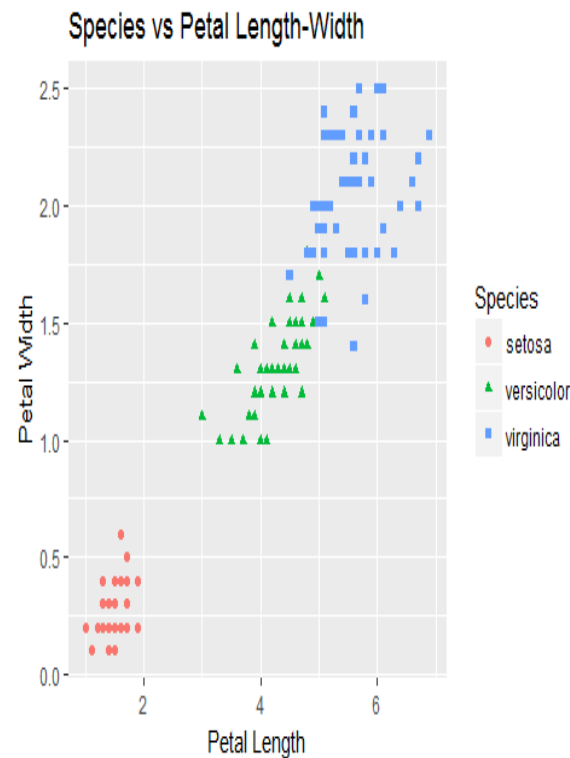
```
#multiplot(scatter1, scatter2, cols=2)
```

We will use the R's KNN to classify the species based on the feature vector.

```
data("iris")
```

```
summary(iris)
```

```
## Sepal.Length Sepal.Width Petal.Length
Petal.Width
## Min. :4.300 Min. :2.000 Min. :1.000 Min.
:0.100
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st
Qu.:0.300
## Median :5.800 Median :3.000 Median :4.350
Median :1.300
## Mean :5.843 Mean :3.057 Mean :3.758
Mean :1.199
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd
Qu.:1.800
## Max. :7.900 Max. :4.400 Max. :6.900 Max.
:2.500
## Species
## setosa :50
## versicolor:50
## virginica :50
##
##
##
```



```
scatter2 <-ggplot(data=iris, aes(x = Sepal.Length, y =
Sepal.Width)) +
geom_point(aes(color=Species, shape=Species)) +
xlab("Sepal Length") +ylab("Sepal Width") +
ggtitle("Species vs Sepal Length-Width")
scatter2
```

```
set.seed(99) # required to reproduce the results
rnum<-sample(rep(1:150)) # randomly generate
numbers from 1 to 150

iris<-iris[rnum,] #randomize "iris" dataset

scaling_function <-function(x)
{
  return ((x-min(x))/(max(x)-min(x)))
}

iris_norm<-
as.data.frame(lapply(iris[,c(1,2,3,4)],scaling_function
))

iris_scaled <-iris_norm;

iris_scaled$Species <-iris$Species

# subset the dataset
iris_norm_train<-iris_norm[1:130,]
iris_norm_train_target<-iris[1:130,5]
iris_norm_test<-iris_norm[131:150,]
iris_norm_test_target<-iris[131:150,5]

# load the classification library that contains knn

require("class")

## Loading required package: class

model1<-knn(train=iris_norm_train,
test=iris_norm_test, cl=iris_norm_train_target, k=5)

table(iris_norm_test_target, model1)

##           model1
## iris_norm_test_target setosa versicolor virginica
##      setosa         5         0         0
##      versicolor      0         8         0
##      virginica       0         2         5

df <-as.data.frame(iris_norm_test_target)

names(df) =c("species")

length(df[df$species == "setosa",])

## [1] 5

length(df[df$species == "versicolor",])

## [1] 8
```

```
length(df[df$species == "virginica",])
```

```
## [1] 7
```

From the above table, we can see out of 20 test examples, 18 were correctly classified and 2 were classified incorrectly.

Now let us simulate KNN manually

We will manually calculate the distances for each test example from all training examples and then find K nearest training examples. Out of these K training examples, we will output the majority class as our guess.

We have already split our scaled data set into training and test subsets. Since the features involved are numerical features, we will manually calculate the distance using Euclidean distance metric. We will calculate the distances for all the training examples from the query example and classify the query example.

Let us first consider the case of setosa, which is the first one in our test set.

Example-1: setosa

Let us take the first test example and generate distances from all examples of the training data set. This is a "setosa". We will find out 5 nearest neighbor and guess the class of our query example.

```
iris_scaled_train<-iris_scaled[1:130,]
iris_scaled_test<-iris_scaled[131:150,]
```

```
iris_scaled_test[1,]
```

```
##      Sepal.Length Sepal.Width Petal.Length
##      Petal.Width Species
## 131  0.1666667  0.4166667  0.06779661
##      0.04166667 setosa
```

```
#131  0.1666667  0.4166667  0.06779661
0.04166667 setosa
```

used the attribute name (instead of index) on test set to make it explicit

```
dists =sqrt((iris_scaled_train[1,] -
iris_scaled_test[1,]$Sepal.Length)^2 +
(iris_scaled_train[2,] -
iris_scaled_test[1,]$Sepal.Width)^2 +
(iris_scaled_train[3,] -
iris_scaled_test[1,]$Petal.Length)^2 +
```

```
(iris_scaled_train[,4] -
iris_scaled_test[1,]$Petal.Width)^2)

df =data.frame(dists,iris_scaled_train[,5][1:130])

names(df) =c("distance","target")

df_ordered <-df[order(df$distance),]

head(df_ordered)

## distance target
## 62 0.04382579 setosa
## 79 0.05007710 setosa
## 19 0.05007710 setosa
## 106 0.06047157 setosa
## 86 0.06131473 setosa
## 60 0.09415490 setosa
```

As we can see above, our query example is classified correctly as "setosa".

Our next query example is a "versicolor". We will find out 5 nearest neighbors from the training set and guess the class of our query example.

Example-2: versicolor

```
iris_scaled_test[3,]

## Sepal.Length Sepal.Width Petal.Length
Petal.Width Species
## 133 0.3333333 0.1666667 0.4745763
0.4166667 versicolor

#133 0.3333333 0.1666667 0.4745763
0.4166667 versicolor

dists =sqrt((iris_scaled_train[,1] -
iris_scaled_test[3,]$Sepal.Length)^2 +
(iris_scaled_train[,2] -
iris_scaled_test[3,]$Sepal.Width)^2 +
(iris_scaled_train[,3] -
iris_scaled_test[3,]$Petal.Length)^2 +
(iris_scaled_train[,4] -
iris_scaled_test[3,]$Petal.Width)^2)

df =data.frame(dists,iris_scaled_train[,5][1:130])

names(df) =c("distance","target")

df_ordered <-df[order(df$distance),]
```

```
head(df_ordered)

## distance target
## 52 0.04498205 versicolor
## 84 0.05286766 versicolor
## 76 0.11980167 versicolor
## 80 0.12951485 versicolor
## 103 0.13792176 versicolor
## 61 0.15682101 versicolor
```

As we can see above, our query example is classified correctly as "versicolor".

Now we will take "virginica" from our test set and determine 5 nearest neighbors and guess the class of our training example.

Example-3: virginica

```
iris_scaled_test[4,]

## Sepal.Length Sepal.Width Petal.Length
Petal.Width Species
## 134 0.5277778 0.5833333 0.7457627
0.9166667 virginica

#134 0.5277778 0.5833333 0.7457627
0.9166667 virginica
```

```
dists =sqrt((iris_scaled_train[,1] -
iris_scaled_test[4,]$Sepal.Length)^2 +
(iris_scaled_train[,2] -
iris_scaled_test[4,]$Sepal.Width)^2 +
(iris_scaled_train[,3] -
iris_scaled_test[4,]$Petal.Length)^2 +
(iris_scaled_train[,4] -
iris_scaled_test[4,]$Petal.Width)^2)

df =data.frame(dists,iris_scaled_train[,5][1:130])

names(df) =c("distance","target")

df_ordered <-df[order(df$distance),]

head(df_ordered)

## distance target
## 109 0.06047157 virginica
## 104 0.10157824 virginica
## 28 0.14069121 virginica
## 48 0.17480315 virginica
## 30 0.17480315 virginica
```

```
## 108 0.17916292 virginica
```

As we can see above, our query example is classified correctly as "virginica".

Choosing the value of K for KNN

As we vary K, our classifications would change [7], because as we increase the value of K, the decision boundary would expand. Hence, the majority class would change also. If we increase the value to a high value close to the size of the training set, it is the majority class in the entire training set. If we choose a very small value for k, for example $k = 1$, noise in the training set would play a very important role [14]. We generally choose an odd number to avoid any tie situations.

```
# subset the dataset
```

```
iris_norm_train<-iris_norm[1:100,]
iris_norm_train_target<-iris[1:100,5]
iris_norm_test<-iris_norm[101:150,]
iris_norm_test_target<-iris[101:150,5]
```

```
# load the classification library that contains knn
```

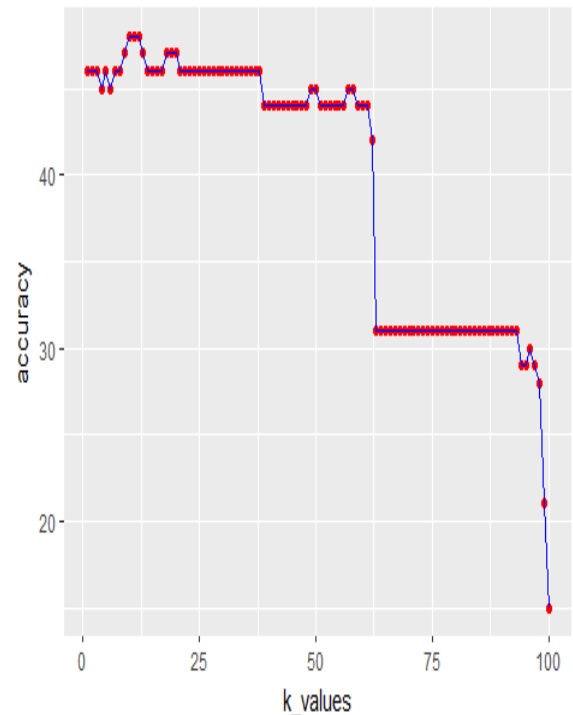
```
k_values =c()
accuracy =c()
require("class")
```

```
for (i in 1:100)
{
  model1<-knn(train=iris_norm_train,
test=iris_norm_test, cl=iris_norm_train_target, k=i)
```

```
#table(iris_norm_test_target, model1)
```

```
k_values[[i]] =i
accuracy[[i]] =sum(diag(table(iris_norm_test_target,
model1)))
#print(paste(i,sum(diag(table(iris_norm_test_target,
model1))))))
}
```

```
data <-data.frame(k_values,accuracy)
ggplot(data)
+geom_point(aes(x=k_values,y=accuracy),col=I("red
")) +
geom_path(aes(x=k_values,y=accuracy),col=I("blue")
) +
labs(caption = "KNN- k-value vs accuracy fit")
```



KNN- k-value vs accuracy fit)

As we can see in the above plot, the optimal value of K is between 10 and 12. Hence, we can take the odd number 11 for K.

Numeric and Categorical Features

The data set we used so far has only numerical attributes. Now let us use a different data set with numerical and categorical features. We will use the data set 'Salaries' supplied with R [13].

```
data("Salaries", package = "car")
```

```
head(Salaries)
```

```
##      rank discipline yrs.since.phd yrs.service sex
salary
## 1   Prof         B          19         18 Male 139750
## 2   Prof         B          20         16 Male 173200
## 3  AsstProf      B           4           3 Male  79750
## 4   Prof         B          45         39 Male 115000
## 5   Prof         B          40         41 Male 141500
## 6  AssocProf    B           6           6 Male  97000
```

```
str(Salaries)
```

```
## 'data.frame':  397 obs. of  6 variables:
## $ rank      : Factor w/ 3 levels
"AsstProf","AssocProf",...: 3 3 1 3 3 2 3 3 3 3 ...
## $ discipline : Factor w/ 2 levels "A","B": 2 2 2 2 2
2 2 2 2 2 ...
```

```
## $ yrs.since.phd: int 19 20 4 45 40 6 30 45 21 18 ...
## $ yrs.service : int 18 16 3 39 41 6 23 45 20 18 ...
## $ sex : Factor w/ 2 levels "Female","Male": 2
2 2 2 2 2 2 2 2 1 ...
## $ salary : int 139750 173200 79750 115000
141500 97000 175000 147765 119250 129000 ...
```

```
unique(Salaries$rank)
```

```
## [1] Prof AsstProf AssocProf
## Levels: AsstProf AssocProf Prof
```

This data set contains numeric and categorical variables. As we can see that rank, discipline and sex are factor variables and others are numeric variables. Here is a complete description of this data set.

Description of the Salaries Data set

This is a data frame with 397 observations on the following 6 variables.

- rank - a factor with levels
 - AssocProf
 - AsstProf
 - Prof
- discipline - a factor with levels A ("theoretical" departments) or B ("applied" departments).
- yrs.since.phd - years since PhD.
- yrs.service - years of service.
- sex - a factor with levels Female Male
- salary - nine-month salary, in dollars.

Salary is a continuous variable, so we can use Linear Regression to create a model. Rank is a categorical variable with three different values, AsstProf, AssocProf and Prof. We will use K-Nearest Neighbors method to predict the rank given the other attributes.

Since some of the features are categorical variables, we will need to encode them before we apply KNN algorithm.

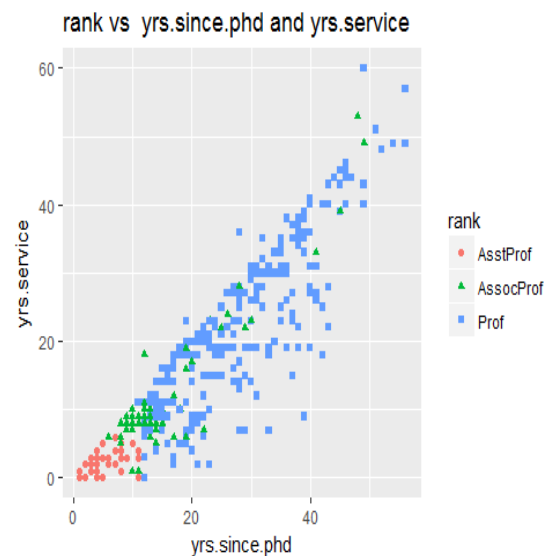
We will use one-hot encoding method to encode our categorical variables discipline and sex. In case of one-hot encoding, we will add new columns based on the number of distinct values to our categorical variable. Since there are two distinct values for the feature discipline we need to generate two different variables discipline_A and discipline_B. The variable discipline_A would take a value of 1 when the original

column value is A and a 0 otherwise. Similarly, the new column discipline_B would take a value of 1 when the original column value is B and 0 otherwise.

Similarly, since there are two distinct values for the feature sex we need to generate two different variables sex_Male and sex_Female. The variable sex_Male would take a value of 1 when the original column value is Male and a 0 otherwise. Similarly the new column sex_Female would take a value of 1 when the original value is Female and 0 otherwise. The following code fragment would achieve it.

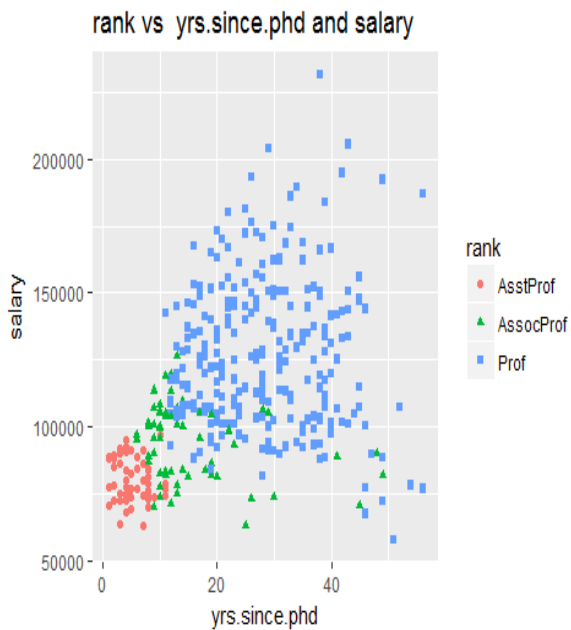
The following code fragment would achieve it. We are weighing them down because the other variables are fractions between 0 and 1.

```
data("Salaries", package = "car")
Salaries$discipline_A <-ifelse(Salaries$discipline
%in%c("A"),1/1000,0)
Salaries$discipline_B <-ifelse(Salaries$discipline
%in%c("B"),1/1000,0)
Salaries$sex_Male <-ifelse(Salaries$sex
%in%c("Male"),1/1000,0)
Salaries$sex_Female <-ifelse(Salaries$sex
%in%c("Female"),1/1000,0)
library(ggplot2)
scatter1 <-ggplot(data=Salaries, aes(x = yrs.since.phd,
y = yrs.service)) +
geom_point(aes(color=rank, shape=rank)) +
xlab("yrs.since.phd") + ylab("yrs.service") +
ggtitle("rank vs yrs.since.phd and yrs.service")
scatter1
```

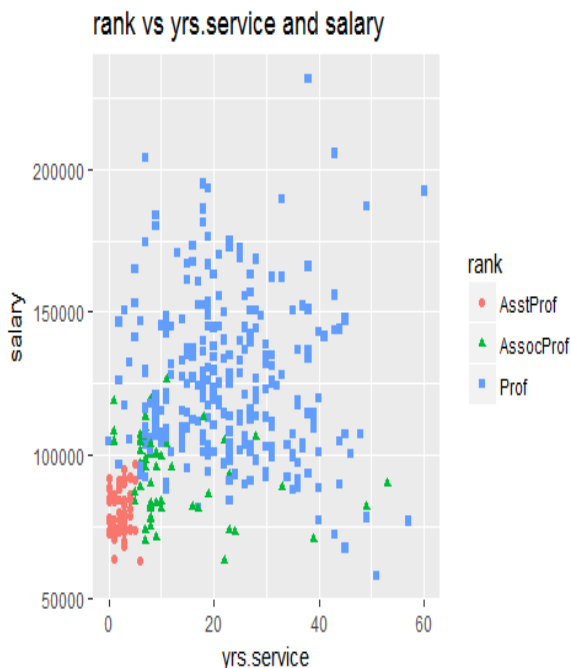


```
scatter2 <-ggplot(data=Salaries, aes(x = yrs.since.phd,
```

```
y = salary)) +
geom_point(aes(color=rank, shape=rank)) +
xlab("yrs.since.phd") + ylab("salary") +
ggtitle("rank vs yrs.since.phd and salary")
scatter2
```



```
scatter3 <- ggplot(data=Salaries, aes(x = yrs.service, y
= salary)) +
geom_point(aes(color=rank, shape=rank)) +
xlab("yrs.service") + ylab("salary") +
ggtitle("rank vs yrs.service and salary")
scatter3
```



```
#multiplot(scatter1, scatter2, scatter3,cols=3)
```

As you can see rank is clearly differentiated by years since PhD and years of service.

Feature Scaling

The salary column ranges from 57800 to 231545, yrs.since.phd column ranges from 1 to 56 and yrs.service ranges from 0 to 60. Since these ranges are different for different features, we need to scale the features so that their ranges are comparable and the distance calculation is not completely influenced by features whose range is high.

```
scaling_function <-function(x){
return ((x-min(x))/(max(x)-min(x)))
}
```

```
Salaries_scaled <-
as.data.frame(Salaries[,c(7,8,9,10)])
```

```
# Scale the numeric features
```

```
Salaries_scaled$yrs.since.phd <-
scaling_function(Salaries$yrs.since.phd)
Salaries_scaled$yrs.service <-
scaling_function(Salaries$yrs.service)
Salaries_scaled$salary <-
scaling_function(Salaries$salary)
```

```
# Finally add the rank column to the scaled data set
```

```
Salaries_scaled$rank <-Salaries$rank
```

Now the ranges are normalized as needed. Let us divide our data set of 397 records into training and test sets.

```
Salaries_scaled_train <-Salaries_scaled[1:350,]
Salaries_scaled_test <-Salaries_scaled[351:397,]
```

Now let us use the KNN algorithm on the standardized data set

```
Salaries_new <-Salaries_scaled[,c(1,2,3,4)]
Salaries_new$yrs.since.phd <-
Salaries_scaled$yrs.since.phd
Salaries_new$yrs.service <-
Salaries_scaled$yrs.service
Salaries_new$salary <-Salaries_scaled$salary
```

```
Salaries_train<-Salaries_new[1:350,]
Salaries_train_target<-Salaries[1:350,]$rank
Salaries_test <-Salaries_new[351:397,]
```

```
Salaries_test_target<-Salaries[351:397,]$rank
```

```
require("class")
```

```
model1<-knn(train=Salaries_train, test=Salaries_test,
cl=Salaries_train_target,k=5,use.all=TRUE)
```

```
table(Salaries_test_target, model1)
```

```
##          model1
## Salaries_test_target AsstProf AssocProf Prof
##      AsstProf      6      0      0
##      AssocProf     1      2      2
##      Prof          0      1     35
```

We will manually compute KNN method to our Salaries dataset and test it for different categories of rank i.e. Prof, AssocProf and AsstProf.

Example-1 - Prof

```
Salaries_scaled_test[1,]
```

```
## discipline_A discipline_B sex_Male sex_Female
yrs.since.phd
## 351      0      0.001 0.001      0      1
## yrs.service salary rank
## 351 0.8166667 0.7433883 Prof
```

```
# discipline_A discipline_B sex_Male sex_Female
yrs.since.phd yrs.service salary rank
# 351      0      1      1      0      1
0.8166667 0.7433883 Prof
```

```
n=1
```

```
dists =abs(Salaries_scaled_train[,1]-
Salaries_scaled_test[n,]$discipline_A) +
abs(Salaries_scaled_train[,2]-
Salaries_scaled_test[n,]$discipline_B) +
abs(Salaries_scaled_train[,3]-
Salaries_scaled_test[n,]$sex_Male) +
abs(Salaries_scaled_train[,4]-
Salaries_scaled_test[n,]$sex_Female) +
sqrt((Salaries_scaled_train[,5] -
Salaries_scaled_test[n,]$yrs.since.phd)^2 +
(Salaries_scaled_train[,6] -
Salaries_scaled_test[n,]$yrs.service)^2 +
(Salaries_scaled_train[,7] -
Salaries_scaled_test[n,]$salary)^2)
```

```
df =data.frame(dists,Salaries[,1][1:350])
```

```
names(df) =c("distance","target")
```

```
df_ordered<-df[order(df$distance),]
```

```
head(df_ordered)
```

```
## distance target
## 331 0.2252499 Prof
## 284 0.2884087 Prof
## 8 0.3087636 Prof
## 72 0.3126065 Prof
## 305 0.3198003 Prof
## 267 0.3586169 Prof
```

Example-2 - AsstProf

```
Salaries_scaled_test[5,]
```

```
## discipline_A discipline_B sex_Male sex_Female
yrs.since.phd
## 355      0      0.001 0.001      0
0.1272727
## yrs.service salary rank
## 355 0.01666667 0.1484935 AsstProf
```

```
# discipline_A discipline_B sex_Male sex_Female
yrs.since.phd yrs.service salary rank
#355      0      1      1      0 0.1272727
0.01666667 0.1484935 AsstProf
```

```
n=5
```

```
dists =abs(Salaries_scaled_train[,1]-
Salaries_scaled_test[n,]$discipline_A) +
abs(Salaries_scaled_train[,2]-
Salaries_scaled_test[n,]$discipline_B) +
abs(Salaries_scaled_train[,3]-
Salaries_scaled_test[n,]$sex_Male) +
abs(Salaries_scaled_train[,4]-
Salaries_scaled_test[n,]$sex_Female) +
sqrt((Salaries_scaled_train[,5] -
Salaries_scaled_test[n,]$yrs.since.phd)^2 +
(Salaries_scaled_train[,6] -
Salaries_scaled_test[n,]$yrs.service)^2 +
(Salaries_scaled_train[,7] -
Salaries_scaled_test[n,]$salary)^2)
```

```
df =data.frame(dists,Salaries[,1][1:350])
```

```
names(df) =c("distance","target")
```

```
df_ordered <-df[order(df$distance),]
```

```
head(df_ordered)
```

```
## distance target
## 32 0.03252605 AsstProf
## 12 0.03296517 AsstProf
## 80 0.04017039 AsstProf
## 275 0.04841544 AsstProf
## 316 0.04974618 AsstProf
## 84 0.05004464 AsstProf
```

Example-3 - AssocProf

```
Salaries_scaled_test[21,]
```

```
## discipline_A discipline_B sex_Male sex_Female
yrs.since.phd
```

```
## 371 0.001 0 0.001 0
0.2181818
```

```
## yrs.service salary rank
## 371 0.1333333 0.1173099 AssocProf
```

```
# discipline_A discipline_B sex_Male sex_Female
yrs.since.phd yrs.service salary rank
#371 1 0 1 0 0.2181818
0.1333333 0.1173099 AssocProf
```

```
n=21
```

```
dists =abs(Salaries_scaled_train[,1]-
Salaries_scaled_test[n,]$discipline_A) +
abs(Salaries_scaled_train[,2]-
Salaries_scaled_test[n,]$discipline_B) +
abs(Salaries_scaled_train[,3]-
Salaries_scaled_test[n,]$sex_Male) +
abs(Salaries_scaled_train[,4]-
Salaries_scaled_test[n,]$sex_Female) +
sqrt((Salaries_scaled_train[,5] -
Salaries_scaled_test[n,]$yrs.since.phd)^2 +
(Salaries_scaled_train[,6] -
Salaries_scaled_test[n,]$yrs.service)^2 +
(Salaries_scaled_train[,7] -
Salaries_scaled_test[n,]$salary)^2)
```

```
head(Salaries_scaled_test)
```

```
## discipline_A discipline_B sex_Male sex_Female
yrs.since.phd
```

```
## 351 0 0.001 0.001 0
1.0000000
```

```
## 352 0 0.001 0.001 0
0.6727273
```

```
## 353 0 0.001 0.001 0
0.4545455
```

```
## 354 0 0.001 0.001 0
0.3818182
```

```
## 355 0 0.001 0.001 0
0.1272727
```

```
## 356 0 0.001 0.001 0
```

```
0.4363636
```

```
## yrs.service salary rank
```

```
## 351 0.81666667 0.7433883 Prof
```

```
## 352 0.63333333 0.2055829 Prof
```

```
## 353 0.45000000 0.4874960 Prof
```

```
## 354 0.33333333 0.4615960 Prof
```

```
## 355 0.01666667 0.1484935 AsstProf
```

```
## 356 0.35000000 0.5020461 Prof
```

```
df =data.frame(dists,Salaries[,1][1:350])
```

```
names(df) =c("distance","target")
```

```
df_ordered <-df[order(df$distance),]
```

```
head(df_ordered)
```

```
## distance target
```

```
## 25 0.02129264 AssocProf
```

```
## 256 0.03315943 AssocProf
```

```
## 109 0.04107320 AssocProf
```

```
## 107 0.04278516 AssocProf
```

```
## 131 0.04867628 AssocProf
```

```
## 317 0.05181494 AssocProf
```

Conclusions:

We have discussed the KNN-algorithm with examples using R. We have discussed the difference between numerical and categorical features. We have seen the importance of feature standardization methods. We have studied different encoding methods for categorical variables. We briefly discussed different distance computation methods for different types of features. We have studied KNN algorithm using R's KNN function on different data sets. Then we have implemented KNN by manually calculating the distances. We have listed the pros and cons of using the KNN algorithm.

References

1. <https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>
2. <https://www.youtube.com/watch?v=PNglugooJUQ&pbjreload=10>
3. <https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e>
4. https://en.wikipedia.org/wiki/Hamming_distance
5. <https://www.datacamp.com/community/tutorials/encoding-methodologies>
6. <https://medium.com/@rahulkuntala9/cosine-similarity-and-handling-categorical-variables-29f907951b5>

7. <https://www.youtube.com/watch?v=9ArwwGaoJwc>
8. <https://www.machinelearningplus.com/nlp/cosine-similarity/>
9. https://en.wikipedia.org/wiki/Chebyshev_distance
10. https://en.wikipedia.org/wiki/Hamming_distance
11. <https://stats.idre.ucla.edu/other/mult-pkg/whatstat/what-is-the-difference-between-categorical-ordinal-and-interval-variables/>
12. <https://www.linkedin.com/pulse/one-hot-encoding-categorical-data-distance-between-english/>
13. <https://vincentarelbundock.github.io/Rdatasets/doc/carData/Salaries.html>
14. <https://discuss.analyticsvidhya.com/t/how-to-choose-the-value-of-k-in-knn-algorithm/2606>
15. https://en.wikipedia.org/wiki/Euclidean_distance
16. <https://www.saedsayad.com/encoding.htm>
17. <https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/>

ijournals