Machine learning - Classification using KNN Model

Stat133 Post 1: Data Technologies

Junfang Jiang

10/30/2017

Machine learning is an area of computer science that gives computer the ability to learn some pattern without explicity programmed. Typical machine learning tasks include Supervised learning (given input and desired output for computer to learn), Unsupervised learning (no labels given, leaving coputer to find the structure of input by itself), and Reinforcement learning (given an environment and a set of rewards or punishments to let computer to reach its goal). The ultimate goal for machine learning is to give the ability to learn and then perform tasks automatically without the interference of human.

In this post, a very basic idea of machine learning will be demonstrated by solving a typical classification question with the well-known 'k-Nearest Neighbors' algorithm.

k-Nearest Neighbors (KNN)

The k-Nearest Neighbour (aka KNN) algorithm is one of the simpliest instance-based machine learning algorithms, in which the new instances are classified by using previous labeled data stored in the database. To be more precise, in KNN algorithm, some "distance" between the new instance and the stored data is calculated and use the distance as an indicator of how likely the new instance belongs the same category as the preivously labeled data. Euclidean distance, cosine similarity or the Manhattan distance are often used to express this similarity distance. (More details on Wikipedia)

Prepare Data

The most important step in all machines learning algorithms is obtaining the right data. Thanks to the R community, there are quite a few builtin data sets in R. For this post, the builtin Iris data set will be used.

1. Overview of the Data

```
head(iris)
```

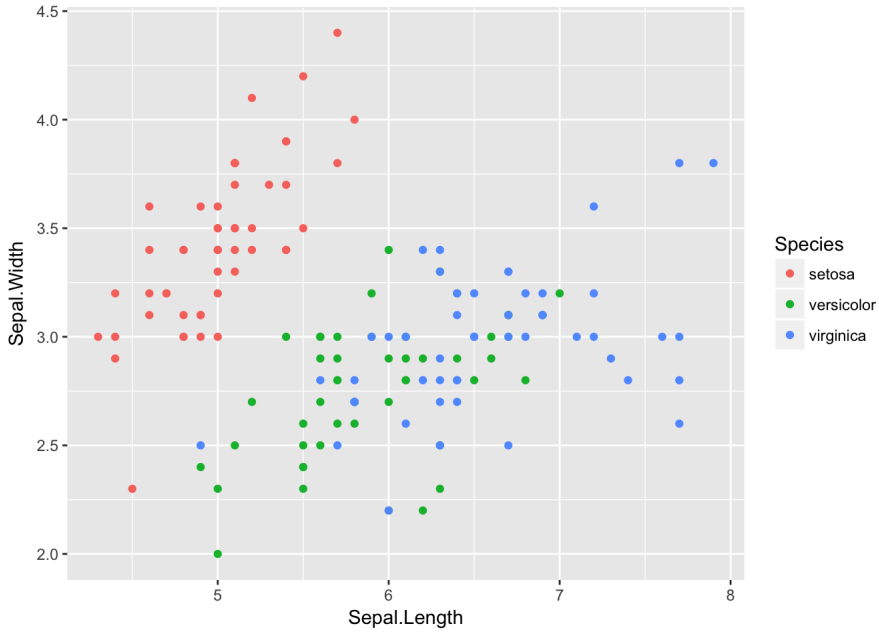
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
summary(iris)
```

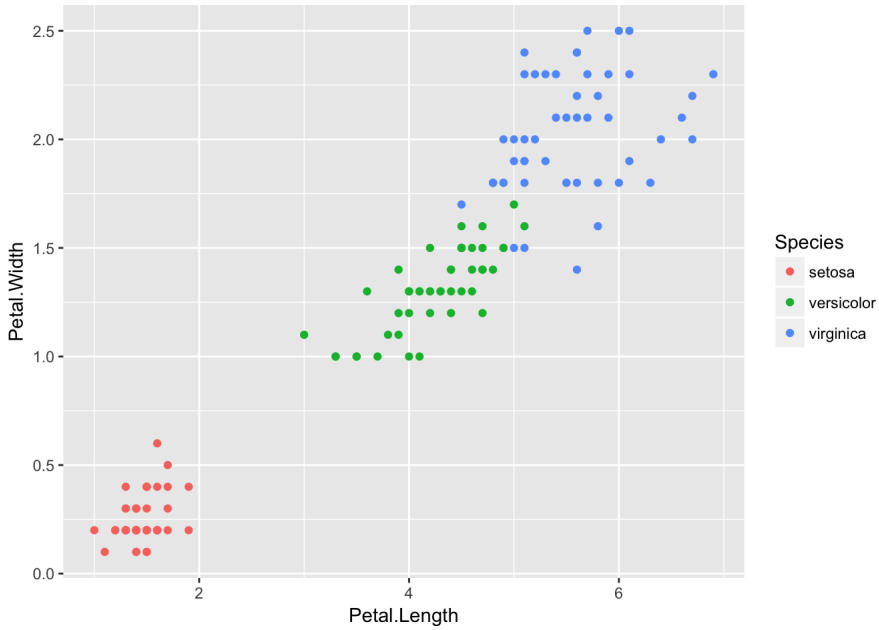
```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   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
##
##
##
```

Iris data set (by Anderson, Edgar, 1935) 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. (More details on Iris)

```
library(ggplot2)
ggplot(iris) + geom_point(aes(x=Sepal.Length, y=Sepal.Width, color=Species))
```




```
ggplot(iris) + geom_point(aes(x=Petal.Length, y=Petal.Width, color=Species))
```



2. Normalize the Data

As a part of data preprocessing, data need to be normalized if different parameters are inconsistent (values have different ranges) with each other. Otherwise, for example, if X has a range of 0 to 1000 while Y has a rnge of 0 to 1, the KNN distance will be dominated by X while the effect of Y is weaken by the power of X.

In Iris data set, it can be observed that `Sepal.Length` has a range from 4.3 to 6.7 while `Sepal.Width` has a range from 2 to 4.4 and `Petal.Width` goes from 0.1 to 2.5. All values are actually within a range of ± 2 to 3 cm, which is considered acceptable, but however, it is never a bad idea to normalize the data.

In the following code, a linear scaling is used for normalization based on min and max value of the individual variables. Other normalizing techniques may also be used.

```
# Normalize function
normalize <- function(x) {
  x_ = x - min(x)
  range = max(x) - min(x)
  return (x_ / range)
}

# Normalize each column of the data set
iris_norm <- as.data.frame(lapply(iris[1:4], normalize))

# Inspect the result
summary(iris_norm)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017   1st Qu.:0.08333
##  Median :0.4167   Median :0.4167   Median :0.5678   Median :0.50000
##  Mean   :0.4287   Mean   :0.4406   Mean   :0.4675   Mean   :0.45806
##  3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949   3rd Qu.:0.70833
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
```

Training and Testing Data Set

In order to evaluate the performance of the model, the data set need to be divided into two parts: training data and testing data. The model will learn the pattern from training data and be tested against the testing data. The point of doing so is because there are always certain degree of error within the data, and the model shouldn't learn on the same data that will be tested on because otherwise, the model might become fitting to the error instead of the actual pattern.

The most common way of dividing data is use 2/3 of data as training data and leave the rest 1/3 as testing data. The data set should be shuffled since there are three different species, and all three species should appear equally likely in both training and testing set. As shown below, a random seed is used for reproducibility and 150 (`nrow(iris)`) random samples are taken between 1 (training) and 2 (testing) where 1 has a probability of 2/3 and 2 has a probability of 1/3.

```
set.seed(80)
index <- sample(2, nrow(iris), replace=TRUE, prob=c(2/3, 1/3))
```

```
# Training Data Set
iris.training    <- iris[index == 1, 1:4]
iris.trainLabels <- iris[index == 1, 5]

# Testing Data Set
iris.test        <- iris[index == 2, 1:4]
iris.testLabels  <- iris[index == 2, 5]

# Inspect the results
head(iris.training)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
## 8          5.0         3.4          1.5         0.2
```

```
print(iris.trainLabels)
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     versicolor versicolor versicolor versicolor
##  [43] versicolor versicolor versicolor versicolor versicolor versicolor
##  [49] versicolor versicolor versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor virginica
##  [73] virginica  virginica  virginica  virginica  virginica  virginica
##  [79] virginica  virginica  virginica  virginica  virginica  virginica
##  [85] virginica  virginica  virginica  virginica  virginica  virginica
##  [91] virginica  virginica  virginica  virginica  virginica  virginica
##  [97] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Use the KNN Model

1. Using `class` package

Now, after all preparation, it is finally time to use KNN and perform the classification. First, by using the easy way, there is a `knn()` function using Euclidian distance implemented in the `class` package (details). By providing our training set, testing set, training label and number of neighbours considered, the new instances in the testing set are classified and stored in `iris_predict` . Here, `k=5` is used which means the 5-nearest neighbour to compute the new label.

```
library(class)
iris_predict <- knn(train = iris.training, test = iris.test, cl = iris.trainLabels, k = 5)
iris_predict
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor versicolor virginica  versicolor versicolor versicolor
## [25] versicolor virginica  versicolor versicolor versicolor virginica
## [31] virginica  versicolor virginica  virginica  virginica  virginica
## [37] virginica  virginica  virginica  virginica  virginica  virginica
## [43] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

2. Implement own `knn` function

We can also implement our own version of KNN algorithm to get a better understanding of how it works.

```
knn <- function(train, test, cl, k) {
  # classify every test instance
  predict = apply(test, 1, function(test){
    # compute squared euclidean distances to all instances in training set
    distance = apply(train, 1, function(x) sum((x - test)^2))
    # find the k-nearest neighbour
    neighbour = order(distance)[1:k]
    # break tie by using the frequency
    frequency = table(cl[neighbour])
    most.frequent = names(frequency)[frequency == max(frequency)]
    # if there are multiple class with same frequency, take a random one
    species = sample(most.frequent, 1)
    return (species)
  })
  return (unname(factor(predict, levels=levels(cl))))
}

iris_predict <- knn(train = iris.training, test = iris.test, cl = iris.trainLabels, k = 5)
iris_predict
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor versicolor virginica  versicolor versicolor versicolor
## [25] versicolor virginica  versicolor versicolor versicolor virginica
## [31] virginica  versicolor virginica  virginica  virginica  virginica
## [37] virginica  virginica  virginica  virginica  virginica  virginica
## [43] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Same result is expected from the two methods and the result is stored in `iris_predict`. Testing labels are not used here because it should be unknown to the computer.

Evaluate the Model

Calculate Percentage Correctness

The next step in machines learning is to evaluate the effectiveness of the model. By comparing the predicted label with the actual label, a percentage of correct classfication can be calculated.

```
# Construct a table to compare predicted label from actual label
compare <- data.frame(iris_predict, iris.testLabels, iris_predict == iris.testLabels)
names(compare) <- c("Predicted Species", "Observed Species", "Matches")

# Calculate the percentage of correct classification
print(sprintf("Accuracy: %f%%", sum(compare$Matches) / length(compare$Matches) * 100))
```

```
## [1] "Accuracy: 93.750000%"
```

```
# Inspect the result`
compare
```

```
##    Predicted Species Observed Species Matches
## 1             setosa           setosa    TRUE
## 2             setosa           setosa    TRUE
## 3             setosa           setosa    TRUE
## 4             setosa           setosa    TRUE
## 5             setosa           setosa    TRUE
## 6             setosa           setosa    TRUE
## 7             setosa           setosa    TRUE
## 8             setosa           setosa    TRUE
## 9             setosa           setosa    TRUE
## 10            setosa           setosa    TRUE
## 11            setosa           setosa    TRUE
## 12            setosa           setosa    TRUE
## 13        versicolor       versicolor    TRUE
## 14        versicolor       versicolor    TRUE
## 15        versicolor       versicolor    TRUE
## 16        versicolor       versicolor    TRUE
## 17        versicolor       versicolor    TRUE
## 18        versicolor       versicolor    TRUE
## 19        versicolor       versicolor    TRUE
## 20        versicolor       versicolor    TRUE
## 21         virginica       versicolor   FALSE
## 22        versicolor       versicolor    TRUE
## 23        versicolor       versicolor    TRUE
## 24        versicolor       versicolor    TRUE
## 25        versicolor       versicolor    TRUE
## 26         virginica       versicolor   FALSE
## 27        versicolor       versicolor    TRUE
## 28        versicolor       versicolor    TRUE
## 29        versicolor       versicolor    TRUE
## 30         virginica        virginica    TRUE
## 31         virginica        virginica    TRUE
## 32        versicolor        virginica   FALSE
## 33         virginica        virginica    TRUE
## 34         virginica        virginica    TRUE
## 35         virginica        virginica    TRUE
## 36         virginica        virginica    TRUE
## 37         virginica        virginica    TRUE
## 38         virginica        virginica    TRUE
## 39         virginica        virginica    TRUE
## 40         virginica        virginica    TRUE
## 41         virginica        virginica    TRUE
## 42         virginica        virginica    TRUE
## 43         virginica        virginica    TRUE
## 44         virginica        virginica    TRUE
## 45         virginica        virginica    TRUE
## 46         virginica        virginica    TRUE
## 47         virginica        virginica    TRUE
## 48         virginica        virginica    TRUE
```

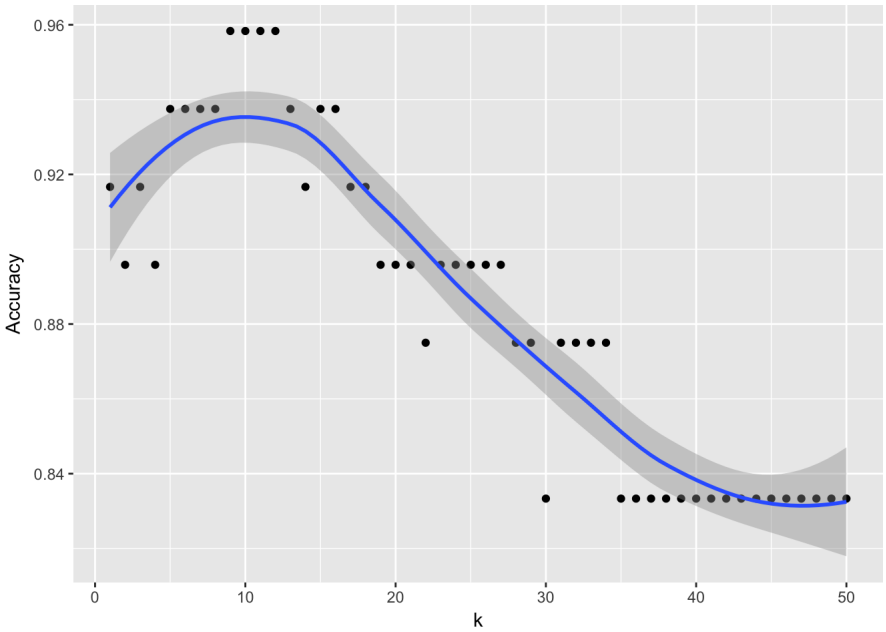
As we can see, the model is making reasonably good predictions with only 3 misclassifications with an accuracy of 93.75%. But what if we want to do better?

Hyperparameter Optimization

In KNN, the number of nearest neighbour (`k`) is a hyperparameter, which means it is a parameter of a prior distribution, instead of parameters of the model for the underlying system under analysis. In previous sections, `k=5` is used while it is possible that other `k` values may produce a better classification.

```r
# Try using different k values from 1 to 50
accuracy = data.frame(k = 1:50, Accuracy = 0)
for (k in accuracy$k) {
  # Run the same KNN algorithm and compute accuracy
  iris_predict <- knn(train = iris.training, test = iris.test, cl = iris.trainLabels, k)
  matches = (iris_predict == iris.testLabels)
  accuracy[k, 2] = sum(matches) / length(matches)
}

# Plot accuracy against k values
ggplot(accuracy) + geom_point(aes(x=k, y=Accuracy)) + geom_smooth(aes(x=k, y=Accuracy), method = "loess")
```



From the diagram above, the model accuracy is maximized at when `k` is somewhere between 8 and 11, which corresponds to an optimal range of the hyperparameter `k`. As a result, by performing the same KNN algorithm again, an correctness rate of 95.8% is acheived.

Last Note

As demonstrated in this post, one of the most attractive features of KNN is that it is simple to understand and easy to implement. It has almost zero training time and can be a useful tool to understand the data before running any other more complex algorithms. However, KNN takes a relatively long time during the test phase, which is not favored by real life applications. Normally, people can accept the fact that machines learning takes a long time for training as the training only need to be done once. When it comes to the time of actually using the model, it should be able to quickly classify or compute result based on new inputs.

Reference

- Wikipedia - k-nearest neighbors algorithm: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- Iris Data Set: http://stat.ethz.ch/R-manual/R-devel/library/datasets/html/iris.html
- IRIS DATASET classification using KNN: https://rpubs.com/Drmadhu/IRISclassification
- Machine Learning in R for beginners: https://www.datacamp.com/community/tutorials/machine-learning-in-r#six