Fast k-Nearest Neighbor Classifier

Contents

Fast Nearest Neighbor Searching 1
The FastKNN Classifier 1
Find the Best k 2
Plot Classification Decision Boundary 3
Benchmark 3

Fast KNN with shrinkage estimator for the class membership probabilities

Fast Nearest Neighbor Searching

The `fastknn` method implements a k-Nearest Neighbor (KNN) classifier based on the ANN library. ANN is written in `C++` and is able to find the k nearest neighbors for every point in a given dataset in `O(N log N)` time. The package RANN provides an easy interface to use ANN library in `R`.

The FastKNN Classifier

The `fastknn` was developed to deal with very large datasets (> 100k rows) and is ideal to Kaggle competitions. It can be about 50x faster then the popular `knn` method from the `R` package class, for large datasets. Moreover, `fastknn` provides a shrinkage estimator to the class membership probabilities, based on the inverse distances of the nearest neighbors (**see the PDF version**):

$$P(x_i \in y_j) = \frac{\sum_{k=1}^{K} \left(\frac{1}{d_{ik}} \cdot (n_{ik} \in y_j) \right)}{\sum_{k=1}^{K} \left(\frac{1}{d_{ik}} \right)}$$

where x_i is the i^{th} test instance, y_j is the j^{th} unique class label, n_{ik} is the k^{th} nearest neighbor of x_i, and d_{ik} is the distance between x_i and n_{ik}. This estimator can be thought of as a weighted voting rule, where those neighbors that are more close to x_i will have more influence on predicting x_i's label.

In general, the weighted estimator provides more **calibrated probabilities** when compared with the traditional estimator based on the label proportions of the nearest neighbors, and reduces **logarithmic loss** (log-loss).

How to install `fastknn`?

The package `fastknn` is not on CRAN, so you need to install it directly from GitHub:

```
library("devtools")
install_github("davpinto/fastknn")
```

Required Packages

The base of `fastknn` is the `RANN` package, but other packages are required to make `fastknn` work properly. All of them are automatically installed when you install the `fastknn`.

- `RANN` for fast nearest neighbors searching,
- `magrittr` to use the pipe operator `%>%`,
- `Metrics` to measure classification performance,
- `ggplot2` to plot classification decision boundaries,
- `viridis` for modern color palletes.

Getting Started

Using `fastknn` is as simple as:

```
## Load packages
library("fastknn")
library("caTools")

## Load toy data
data("chess", package = "fastknn")

## Split data for training and test
tr.idx <- caTools::sample.split(Y = chess$y, SplitRatio = 0.7)
x.tr   <- chess$x[tr.idx, ]
x.te   <- chess$x[-tr.idx, ]
y.tr   <- chess$y[tr.idx]
y.te   <- chess$y[-tr.idx]

## Fit KNN
yhat <- fastknn(x.tr, y.tr, x.te, k = 10)

## Evaluate model on test set
sprintf("Accuracy: %.2f", 100 * sum(yhat$class == y.te) / length(y.te))
```

```
## [1] "Accuracy: 99.55"
```

Find the Best k

The `fastknn` provides a interface to select the best `k` using n-fold cross-validation. There 4 possible **loss functions**:

- Overall classification error rate: `eval.metric = "overall_error"`
- Mean in-class classification error rate: `eval.metric = "mean_error"`
- Mean in-class AUC: `eval.metric = "auc"`
- Cross-entropy / logarithmic loss: `eval.metric = "logloss"`

```
cv.out <- fastknnCV(chess$x, chess$y, k = 3:10, folds = 5, eval.metric = "logloss")
cv.out$cv_table
```

fold_1	fold_2	fold_3	fold_4	fold_5	mean	k
0.2054	0.1105	0.1102	0.1063	0.01642	0.1098	3
0.04497	0.03585	0.02962	0.02453	0.02061	0.03112	4
0.04487	0.03758	0.02911	0.02677	0.0209	0.03185	5
0.04809	0.03779	0.03113	0.0271	0.02282	0.03339	6
0.04795	0.04004	0.03148	0.02856	0.02343	0.03429	7
0.0506	0.03988	0.03532	0.02916	0.02468	0.03593	8
0.05159	0.04139	0.03554	0.03264	0.0249	0.03721	9
0.0535	0.04359	0.03801	0.03443	0.02754	0.03942	10

Plot Classification Decision Boundary

Benchmark