



k -NN Model Tuning and Optimization

An important aspect of any model building is the concept of model tuning. That is, determining the values of the model parameters for optimal results. In the case of k -NN, that parameter is k , the number of nearest neighbors to consider.

Parameter versus Hyperparameter

- A model parameter is a configuration variable that is internal to the model and whose value can be estimated from data.
- Model parameters:
 - are required by the model when making predictions.
 - are estimated or learned from data.
 - are often not set manually by the practitioner.
 - are often saved as part of the learned model.
- Example: Coefficients of variables in linear regression models.

A subtle, but important distinction, exists between parameters for a model that are estimated directly from data, like those of the coefficients of linear regression, and those that are determined outside of the data. For example, the parameters in linear regression are the coefficients of Y-intercept, and slope for the variables. These values are estimated directly from data. On the other hand, the number of neighbors k used in k -NN is not estimated from the data. This is a value chosen by the modeler, and while the optimal value is obviously influenced by the data, the value of k is not estimated from the data. When we talk about model tuning, we are talking about parameters like k , which we call hyperparameters.

Parameters versus hyperparameter continued

- A model hyperparameter, on the other hand, is a configuration that is external to the model and whose value cannot be estimated from data.
- Choosing right hyperparameters can significantly improve the performance of a model.
- Example of hyperparameter: the number neighbours, k , in a k-NN model.
- It is a common practice to select a model's hyperparameters by trying different values and evaluating the performance (i.e. grid search) on validation set (e.g. cross validation)

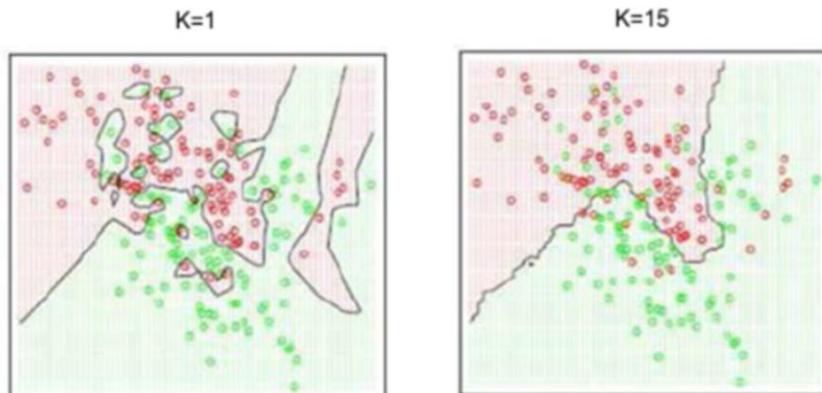
One approach to determining a good value for hyperparameters is to use a validation set. As our primary purpose in many cases is for prediction, this approach allows us to determine what value of the parameter allows the best prediction. The code given in github for this chapter shows an example of this. For us, in k-NN, the hyperparameter we are interested in is k , the number of nearest neighbors to consider.

Effect of k in the k -NN classifier

- If k is chosen to be too small, the model may overfit and the noise and outliers may significantly affect the predictions.
- On the other hand, if k is chosen to be too large, some of the neighbors may be irrelevant to the prediction. This normally results in under fitting.
- The optimal value of k depends on the classification task and the data and should be determined by the modeler in advance.

So, how do we choose k , and what are the effects of the different values of k ?

The advantage of choosing $k > 1$ is that higher values of k provide smoothing that reduces the risk of overfitting due to noise in the training data. Generally speaking, if k is too low, we may be fitting to the noise in the data. However, if k is too high, we will miss out on the method's ability to capture the local structure in the data, one of its main advantages. In the extreme, $k = n$ = the number of records in the training dataset. In that case, we simply assign all records to the majority class in the training data, irrespective of the values of independent variables, which coincides with the naive rule! This is clearly a case of oversmoothing in the absence of useful information in the predictors about the class membership. In other words, we want to balance between overfitting to the predictor information and ignoring this information completely. A balanced choice greatly depends on the nature of the data. The more complex and irregular the structure of the data, the lower the optimum value of k . Typically, values of k fall in the range 1 to 20. We should use odd numbers to avoid ties.

Effect of k in the k-NN classifier continued

See the effect on the classifier as we vary the value of k . For $k=1$, the model is completely responsive to the neighbor, and the classifier responds to that, while for $k=15$, the classifier is more smooth, but not as responsive.

Caret for Hyperparameter Optimization

- Caret package provides a wrapper for a large number of machine learning models (more than 200 models as of now)
- The [train\(\)](#) function will automatically perform a grid search over a pre-defined set of hyperparameter values. It then select the best hyperparameter values using cross validations and train a model.
- The hyperparameter for the k -NN models is k
- The output shows the performance of the model for different k values. Accuracy is used as the default metric to choose the best model.

We will now use the Caret package, and specifically the `train()` function to automatically do a grid search. This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure. Click on the link in the slide to know more about this function. The next several slides illustrates this function. The code is available as part of our github account for the class.

Example: Tuning k

```
library(ISLR)
library(caret)
#Default dataset is a part of the ISLR package
# It captures credit card default status as well as balance, income, and student status
summary(Default)
## default      student      balance      income
## No :9667      No :7056      Min.   : 0.0      Min.   : 772
## Yes: 333      Yes:2944      1st Qu.: 481.7    1st Qu.:21340
##                                     Median : 823.6    Median :34553
##                                     Mean   : 835.4    Mean   :33517
##                                     3rd Qu.:1166.3   3rd Qu.:43808
##                                     Max.   :2654.3    Max.   :73554

#Let's normalize the data before modelling
norm_model<-preProcess(Default, method = c('range'))
Default_normalized<-predict(norm_model,Default)
summary(Default_normalized)
## default      student      balance      income
## No :9667      No :7056      Min.   :0.0000    Min.   :0.0000
## Yes: 333      Yes:2944      1st Qu.:0.1815    1st Qu.:0.2826
##                                     Median :0.3103    Median :0.4641
##                                     Mean   :0.3147    Mean   :0.4499
##                                     3rd Qu.:0.4394    3rd Qu.:0.5913
##                                     Max.   :1.0000    Max.   :1.0000
```

The normalization technique illustrated here is range. We could as easily use the center and scale methods to perform the z-normalization on this data.

Example: Tuning *k* continued

```
#Let's train a k-NN model using the train() function from Caret
# By setting the random seed, we can reproduce the results
set.seed(123)
model<-train(default~balance+income, data=Default_normalized, method="knn")
model

## k-Nearest Neighbors
##
## 10000 samples
##    2 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, ...
## Resampling results across tuning parameters:
##
## k  Accuracy  Kappa
## 5  0.9642461  0.3497580
## 7  0.9668646  0.3648712
## 9  0.9691627  0.3841728
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

The `train()` function provides many methods for regression and classification. *k*-NN is only one such method. We use the normalized data, and *k*-NN as the method here. Note that cross-validation is automatically done by the `train()` function, and here it uses Bootstrapping as the resampling method. Bootstrapping is a way of taking samples with replacement from the underlying data, and is one of the effective methods for resampling for our purposes. By default, for classification, the `train()` function uses Accuracy, the amount correctly classified, to determine the best *k*, but other metrics of performance can also be specified. All aspects of the `train()` function can be controlled using the `trainControl()` function in R. See `?train()` for more information.

Customizing the Search Grid

```
set.seed(123)
Serach_grid <- expand.grid(k=c(2,7,9,15))
model<-train(default=balance+income, data=Default_normalized,
              method="knn", tuneGrid=Serach_grid)

model

## k-Nearest Neighbors
##
## 10000 samples
##    2 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, 10000, ...
## Resampling results across tuning parameters:
##
##    k  Accuracy  Kappa
##    2  0.9562678  0.3005838
##    7  0.9667990  0.3638844
##    9  0.9690868  0.3829013
##   15  0.9714312  0.3971779
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

We now add the tuneGrid option to the function by setting up distinct values to use for finding optimal k.

Pre-Process (e.g. normalize) and Train in one Function

```
set.seed(123)
Serach_grid <- expand.grid(k=c(2,7,9,15))
model<-train(default~balance+income, data=Default,
             method="knn", tuneGrid=Serach_grid,
             preProcess="range");

model

## k-Nearest Neighbors
## 10000 samples
## 2 predictor
## 2 classes: 'No', 'Yes'
## Pre-processing: re-scaling to [0, 1] (2)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, ...
## Resampling results across tuning parameters:
##
##  k  Accuracy  Kappa
##  2  0.9561052  0.3002330
##  7  0.9669296  0.3675515
##  9  0.9689893  0.3811722
## 15  0.9713991  0.3964302
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

Finally, we incorporate all aspect, including search grid, normalization, and method into the function.