K Nearest Neighbors

36-290 – Statistical Research Methodology

Week 9 Tuesday – Fall 2021

KNN

In words: KNN examines the k data points closest to a location x and uses just those data to generate predictions. The optimal value of k is that which minimizes validation-set MSE (regression) or, e.g., MCR (classification).

KNN straddles the boundary between fully parameterized models like linear regression and fully data-driven models like random forest. A KNN model is data-driven, but one *can* actually write down a compact parametric form for the model *a priori*:

- For regression:

$$\hat{Y}|\mathbf{x} = \frac{1}{k}\sum_{i=1}^{k} Y_i \,,$$

  where the summation is over the points $\mathbf{x}_1, \ldots, \mathbf{x}_k$ that are "closest" to $\mathbf{x}$ (usually in a Euclidean sense).

- Classification involves the same summation, with a twist:

$$P[Y = j|\mathbf{x}] = \frac{1}{k}\sum_{i=1}^{k} \mathbb{I}(Y_i = j) \,,$$

  where $\mathbb{I}(\cdot)$ is the indicator function. It returns 0 if the argument is false, and 1 otherwise. The summation yields the proportion of neighbors that are of class j.

Finding the Optimal Number of Neighbors

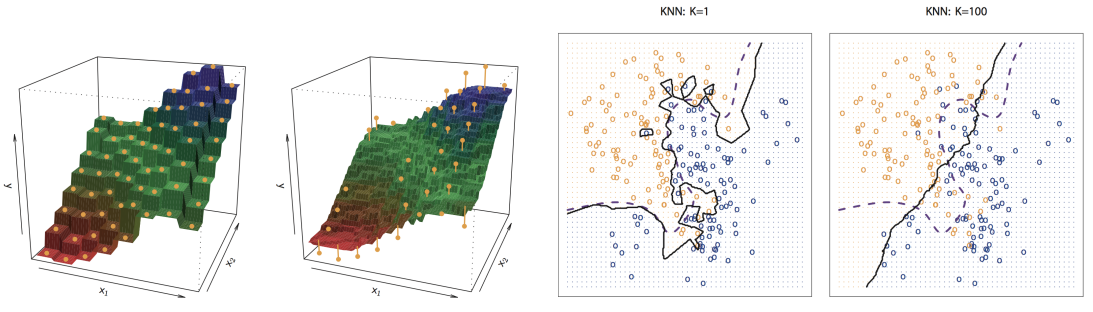
For KNN, the number of neighbors k is a *tuning parameter* (like λ is for lasso). This means that in addition to splitting data into training and test datasets, *the training data themselves have to be split into a smaller training set and a validation set*. For each value of k, you train on the smaller training set, and compute MSE or MCR using the validation set. Once the optimum value of k is determined, you re-run the model using the *full* (unsplit) training set, and assess the model using the test data. (Again: the test data are never used when learning/tuning the model!)

(Note, however: the `FNN` package in `R` does cross-validation on the training set for you, under the hood...you do not need to code this yourself! I mention what I mention above because you should be aware that any algorithm with tunable parameters needs to utilize a validation set [or do cross-validation on the training set] in order to work.)

Finding the Optimal Number of Neighbors

As is the case elsewhere in statistical learning, determining the optimal value of k requires balancing bias and variance:

- If k is too small, the resulting model is *too flexible*, which means it has low bias (it is right on average) but high variance (the predictions are more uncertain). See the panels to the left below.
- If k is too large, the resulting model is *not flexible enough*, resulting in high bias (wrong on average) and low variance (nearly the same predictions, every time). See the panels to the right below.



(Figures 3.16 and 2.16, *Introduction to Statistical Learning* by James et al.)

KNN in Context

Here are two quotes from ISLR to keep in mind when thinking about KNN:

- "As a general rule, parametric methods [like linear regression] will tend to outperform non-parametric approaches [like KNN] when there is a small number of observations per predictor." This is the *curse of dimensionality*: for data-driven models, the amount of data you need to get similar model performance goes up exponentially with p.

$\Rightarrow$ KNN might not be a good model to learn when the number of predictor variables is large.

- "Even in problems in which the dimension is small, we might prefer linear regression to KNN from an interpretability standpoint. If the test MSE of KNN is only slightly lower than that of linear regression, we might be willing to forego a little bit of prediction accuracy for the sake of a simple model..."

$\Rightarrow$ KNN is not the best model to learn if inference is the goal of an analysis.

KNN: Three Critical Points to Remember

- Since it utilizes distances between data, KNN only works with quantitative predictor variables.
- To determine which neighbors are the nearest neighbors, pairwise Euclidean distances are computed...so we need to scale (or standardize) the individual predictor variables so that the distances are not skewed by that one predictor that has the largest variance.
- Don't blindly compute a pairwise distance matrix! For instance, if n = 100,000, then your pairwise distance matrix will have 10^{10} elements, each of which uses 8 bytes in memory...resulting in a memory usage of 80 GB! Your laptop cannot handle this. It can barely handle 1-2 GB at this point. If n is large, you have three options:
    - subsample your data, limiting n to be $\lesssim$ 15,000-20,000;
    - use a variant of KNN that works with sparse matrices (matrices that can be compressed since most values are zero); or
    - make use of a "kd tree" to more effectively (but only approximately) identify nearest neighbors.

The `FNN` package in `R` has an option to search for neighbors via the use of a kd tree.

KNN: Regression Example

We use the same 10,000-galaxy dataset that we used to illustrate random forest. However, here, we scale the predictors.

```
library(FNN)

k.max = 50
mse.k = rep(NA,k.max)
for ( kk in 1:k.max ) {
  knn.out = knn.reg(train=pred.train,y=resp.train,k=kk,algorithm="brute")
  mse.k[kk] = mean((knn.out$pred-resp.train)^2)
}
k.min = which.min(mse.k)
cat("The optimal number of nearest neighbors is ",k.min,"\n")
```

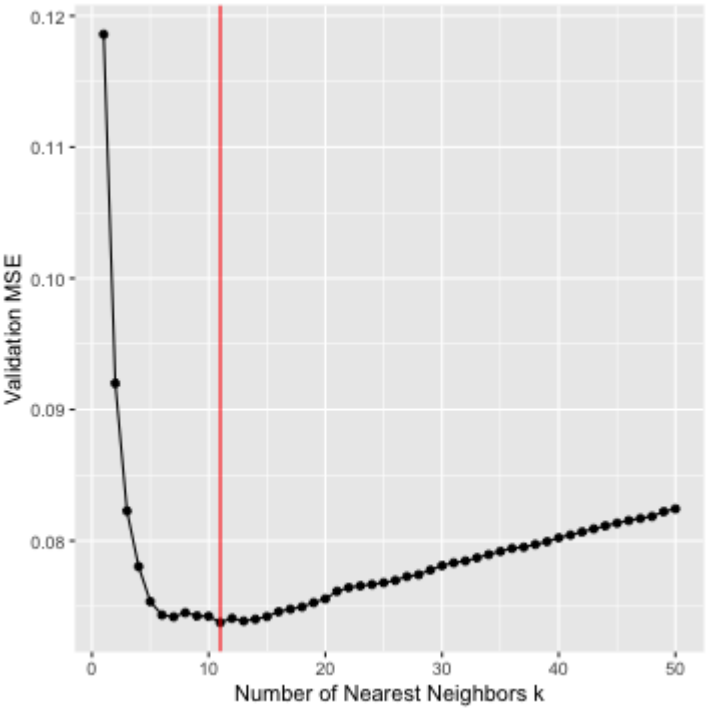
```
## The optimal number of nearest neighbors is  11
```

Note: if the optimal number of nearest neighbors is $k_{\max}$, then your value of $k_{\max}$ is too small. Increase it and start over!

KNN: Regression Example

```
suppressMessages(library(tidyverse))

ggplot(data=data.frame("k"=1:k.max,"mse"=mse.k),mapping=aes(x=k,y=mse)) +
  geom_point() + geom_line() +
  xlab("Number of Nearest Neighbors k") + ylab("Validation MSE") +
  geom_vline(xintercept=k.min,color="red")
```

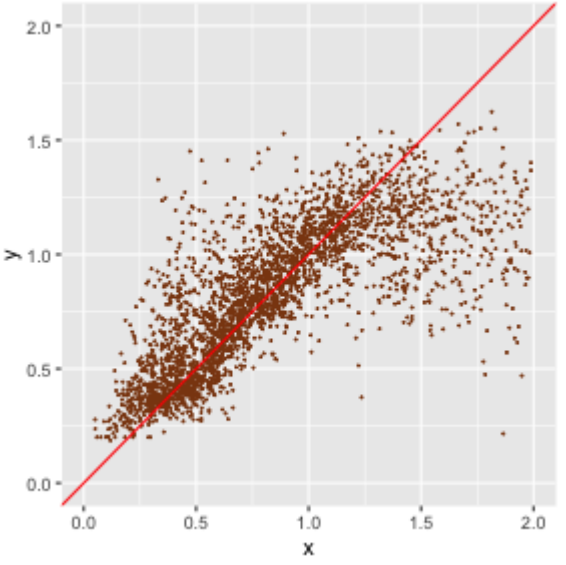


KNN: Regression Example

```
knn.out = knn.reg(train=pred.train,test=pred.test,y=resp.train,k=k.min,algorithm="brute")
(knn.mse = mean((knn.out$pred-resp.test)^2))
```

```
## [1] 0.07574588
```

```
ggplot(data=data.frame("x"=resp.test,"y"=knn.out$pred),mapping=aes(x=x,y=y)) +
  geom_point(size=0.1,color="saddlebrown") + xlim(0,2) + ylim(0,2) +
  geom_abline(intercept=0,slope=1,color="red")
```



KNN: Classification Example

Note that for classification:

- instead of using `knn.reg()`, you would use `knn.cv()` to determine the optimum value of k; and
- you would use `knn()` to generate predictions on the test set.

Note the `cl` argument for both: this is where `resp.train` goes (since `cl` means "class").

`knn()` will output class predictions (based on majority vote!), unless you add the argument `prob=TRUE` to the call. If you want to compute, e.g., a ROC curve for a KNN classifier, use the (pseudo-)code below to extract the Class 1 probabilities. Once you have those, you can create the ROC curve, or a confusion matrix for a specific class-separation threshold, etc., etc.

```
knn.out = knn(...,prob=TRUE)
knn.prob = attributes(knn.out)$prob
w = which(knn.out=="<class 0>") # insert name of Class 0 here
knn.prob[w] = 1 - knn.prob[w]   # knn.prob is now the Class 1 probability!
```