Problem Set #3

Kevin McAlister

2023-01-26

This is the third problem set for QTM 347. This homework covers an applied exercise related to validation sets and K-nearest neighbors regression.

Please use the intro to RMarkdown posted in the Intro module and my .Rmd file as a guide for writing up your answers. You can use any language you want.

Your final submission on Canvas should include, at a minimum, 2 files: 1) a .Rmd/.ipynb file and 2) either a rendered HTML file or a PDF.

This assignment is due by February 3rd, 2023 at 11:59 PM EST.

`TaiwanPricing_TrainS2023.csv` and `TaiwanPricing_TestS2023.csv` are data sets that include information about real estate valuation in Sindian District, New Taipei City, Taiwan. The training data has 314 instances of property value (`Price`) in New Taiwan Dollars per 3.3 meters squared, observation IDs (`ID`), and 7 predictors:

1. `Year` and `Month` of transaction
2. `Age` of property sold
3. `DisttoMRT`: The distance from the property to the nearest Taiwan Metro Transit station
4. `NumConvStore`: The number of convenience stores in the living circle
5. `Lat` and `Long`: Latitude and Longitude of the property

Your goal for this problem set is to build a model that does a good job of predicting the price of properties that are not included in the training data set using only latitude and longitude as predictors.

An important note: The test data set with 50 observations is our holdout data for assessing the quality of our predictions on data that have no part in training the model. Do not touch this data when training your model! You can forget about this data set until the final part of this assignment.

Problem 1 (100 pts)

Part 1 (10 pts)

For the training data, create a plot that shows the relationship between the latitude and longitude of the property and its price. There are many ways to accomplish this: you can try creating a 3D scatter plot, but

I think there are more clever ways to accomplish this in 2 dimensions using color, point size, or smoothers to create a contour plot.

What relationships do you see? Do you think that this relationship is easily captured using a linear model? Why or why not?

Part 1 Solution

Part 2 (15 pts)

This is an example of a relationship with complicated dependency patterns - the latitude and longitude work together to predict the value of a home. In this case, there is a city centroid where prices tend to be high and prices decrease in different ways as the home gets further from the centroid. Because of the changing relationship, the standard linear model is unlikely to do too good of a job of uncovering the underlying price generating function!

One approach to capturing this kind of relationship is to move to a **nonparametric model** that gives us an opportunity to capture the weird non-standard dependence structures that come from geographic data. We haven't discussed many of these yet, but we have briefly discussed K-nearest neighbors regression. For this kind of problem where *distance* is actually distance, this simple model makes a lot of sense.

As a reminder, K-nearest neighbors regression is a voting-style model that maps outcomes from the training data to predictions using the following algorithm:

1. Given a vector $\boldsymbol{x}_{Test}$ in the P-dimensional covariate space, compute the distance (however you may want to define distance) from $\boldsymbol{x}_{Test}$ to each $\boldsymbol{x}_{i,Train}$ in the training set.
2. Select the K smallest distances and store the set of k outcomes, $y_{i,Train}$.
3. Compute y_{Test} as a distance-weighted combination of the k selected outcomes:

$$y_{Test} = \sum_{k=1}^{K} \alpha_k y_{k,Train}$$

where α_k is a weight that is a function of the distance between the test point and the respective training point (smaller distance goes with larger weight). Sometimes, this weight is just set to $\frac{1}{K}$.

At its core K-nearest neighbors is a pretty simple algorithm, but the search for nearest neighbors can be quite intensive with hacky approaches (like the kinds we would write if we were to naively write the search algorithm). For this reason, we'll use prebuilt implementations for our analyses. In R, I recommend using the function `knn.reg` in the `FNN` package or `knnreg` in the `caret` package. In Python, the `KNeighborsRegressor` function in `sklearn` will do the same thing.

Using the latitude, longitude, and price included in the training data set, compute the mean squared error of the predictions made by the model **for the in-sample data** for K-nearest neighbors for 1 through 20 nearest neighbors. Plot the mean squared error (MSE) against the number of nearest neighbors.

Recall that the in-sample MSE always decreases as the flexibility and complexity of the model increases. What is the least complex value of k for the K-nearest neighbors algorithm? In a sentence or two, explain why this is the case.

Part 2 Solution

Part 3 (30 pts)

Write a function called `knn_reg_kfold` that takes five arguments: 1) `x` - a $N \times P$ matrix of predictors, 2) `y` - a N vector of outcomes, 3) `k` - an integer value for the number of neighbors to take, 4) `folds` - an integer

value for the number of folds in the data, and 5) `seed` - an integer value that will define the seed that dictates how the folds are divided. Your function should return the K-fold cross validation estimate of the expected mean squared prediction error for a new observation.

The implementation of K-nearest neighbors that you choose may have a built in K-fold cross validation method. For this problem, please construct the folds and estimates yourself!

Using `knn_reg_kfold`, set the number of folds to 2,5,10,20, and N (LOOCV) and estimate the cross validation estimate of the mean squared prediction error. Plot your estimate of the expected prediction error against the number of nearest neighbors for each number of folds on the same graph. Does the number of neighbors with the lowest K-fold prediction error remain the same/similar across all the number of folds?

Note: Leave one out cross validation (or setting `folds = N`) should be relatively quick given this data size and the speed of the KNN algorithms in the recommended packages. If you find that your function is really dragging, check that your implementation is using a compiled sort and search. If that doesn't work, reach out and I can try to provide some advice.

Hints:

First, please explicitly set a seed at the beginning of your function! The `seed` argument should be passed as the random seed. In R, you should use `set.seed(seed)` to set the seed to the passed value.

Creating the folds can cause problems if you aren't careful in the implementation. Some steps to follow if you're struggling:

1. Find the number of observations in the training set, N
2. For the number of folds, F, find the minimum integer number of observations per fold, N_F, such that:

$$N_F^* = \underset{N_F \in \mathbb{N}}{\text{argmin}}\ F \times N_F \geq N$$

   This is an excellent place to use the **ceiling** function as the solution is $\lceil N/F \rceil$
3. Create a sequence of integer values from 1 to F and repeat it N_F^* times
4. Subset the repeated integer vector to only include the first N elements
5. Randomly shuffle the N-vector of integer fold labels!

Once you have the shuffled labels, computing the cross-validation estimator of the EPE is as simple as setting up a for loop and **subsetting** the train and test data in each iteration.

Part 3 Solution

Part 4 (10 pts)

Create the plot in Part 2 again using different seeds for your validation set splits. Do the results change? What does this say about the relationship between the estimated expected prediction error and your choice of splits?

Part 4 Solution

Part 5 (25 pts.)

Using the above information, choose a value of K as the "optimal" choice. In a few sentences, defend your choice. You can use any heuristic you'd like to make this choice.

Using your chosen value of K, create a latitude-longitude prediction map - using the training data, draw the predictions for new observations within the square created by the minimum and maximum latititudes and longitudes. Create the same drawings for a 1-nearest neighbor regression and a 20-nearest neighbor regression. Compare these pictures to your original plot. How do the different choices compare to the training data?

Hints:

1. There are a number of different ways to create this plot. I'd recommend using a grid-based approach to create a set of predictions and then a smoothed contour plot or colored dot plot to demonstrate the relationship between lat, long, and price.
2. In 2-dimensions, we don't need a huge grid set. 150 equally spaced points per axis should be plenty.
3. Suppose we have two sequences of values and we want a data frame with all possible pairwise combinations of the values. In R, check out the `expand.grid()` function. This way, you can pass this as the test data for a KNN regression and only have to do the search once (as opposed to a double for loop). In Python, check out `meshgrid` in `numpy` - it can quickly be converted to a `numpy` or `Pandas` array.

An example plot:

Part 5 Solution

Part 6 (10 pts.)

Alas, we've judged out-of-sample predictive power without actually ever using an out-of-sample data set - a.k.a. the "original sin". Fortunately, your gracious teacher has held out some test data for you.

Using the **entire** training data as the training values, create predictions for the test data set setting the number of nearest neighbors to your optimal choice. Use these predictions to compute a true EPE estimate for the test set.

How does the holdout MSE value for the test data set compare to the EPE estimate produced using cross-validation?

Note: The test data set in this case is randomly drawn from the original data set. Your final model is also quite simple - we have two features and a reasonable number of nearest neighbors such that the model isn't too complex. This means that the size of the training set is much larger than the relative complexity of the fitting procedure. Similarly, your final model fit uses all 314 data points rather than the subsets used in each fold of the cross-validation procedure. In this case, we often see that good CV estimates **overestimate** the expected prediction error! I think this phenomenon is well-explained by this StackOverflow answer.

Part 6 Solution

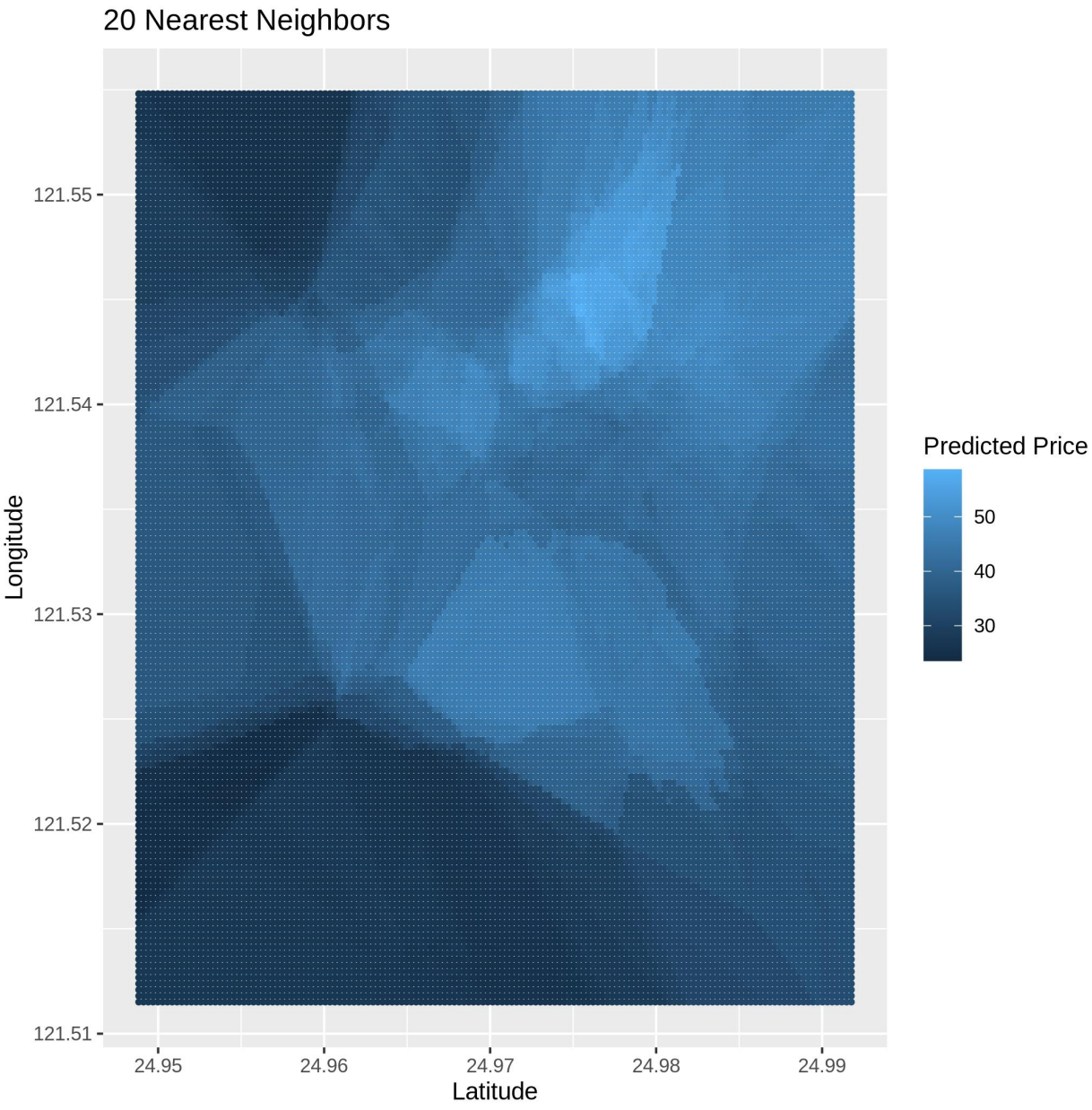

Figure 1: Example