Doug Hummel-Price

November 16th, 2019

**Machine Learning Tutorial: k-nearest neighbors[[1]](#footnote-1)**

The continually growing data science landscape contains many machine learning tools. Learning how to interpret and validate the results of these tools can prove quite difficult. This post will explore the logic behind the model "K-nearest neighbors", or KNN. We will first discuss KNN at a high level and then apply the model to a NASA dataset, attempting to predict whether a given state has a Senator sitting on the Senate subcommittee responsible for NASA appropriations.

*Note: Explaining every aspect of a certain machine learning model would fill a book. This tutorial presumes that the reader understands what it means to map variables to points in an n-dimensional space and measure the distance between these points.[[2]](#footnote-2) This tutorial also presumes that the reader understands the reason behind a training/test split of data in a statistical learning model.[[3]](#footnote-3)*

**An Introduction to KNN**

KNN as we will use here is a classifier, meaning that it takes a data set with many *features* (aka independent variables) and a single categorical *target* (aka dependent variable). KNN is appropriate when the target is discrete, regardless of whether that target is ordinal. There is no strict maximum number of categories for the target. However, the accuracy of the KNN model, all else equal, requires more data for each additional category, so KNN is typically used for a small number of categories. KNN works best when the features are continuous quantitative variables, though dummy variables can be used to turn categorical variables into quantitative ones.

The Logic Behind KNN

Suppose we are on a university campus in which students have a variety of dorms to pick from. Not surprisingly, students with similar interests choose to live together. Athletes may decide to live in one dorm while artists opt to live in another. Now suppose we have data on each student: their extracurriculars, grades and coursework, major, gender, home state, and similar information that might be relevant to where that student might choose to live, in addition to the actual dorm that student chose.

Now suppose we have the same information about the next year of students who have yet to pick their dorms. We are tasked with predicting where these students will live to ensure that no single dorm will be overpopulated. To do so, we will analyze each new student and attempt to find a set number of students from the previous year with similar characteristics.

Let's make this more tangible:

A screenshot of a cell phone

Description automatically generated

Suppose that the school has 37 total students, 20 artists and 17 athletes. Also suppose the school only has two dorms, the Athletic dorm and the Artistic dorm. In the Athletic dorm, 14 students are active athletes, and the remaining 3 are artists who prefer the atmosphere of the Athletic dorm to that of the Artistic dorm. The Artistic dorm has 17 active artists and 3 athletes who prefers the Artistic dorm. Due to their rigorous schedules, most athletes take a predictable set of big lecture courses and receive GPAs high enough to keep them NCAA-eligible. Artists take a separate, though similarly predictable set of small seminars and have a much higher variety of GPA, both high and low. We have plotted them on the scatterplot below.

We are then presented with a new student. This student takes mostly small seminars and has an average GPA. Let's add this student to our plot.

A screenshot of a cell phone

Description automatically generated

The *k*-parameter

In the above example, the number of students that we examine similar to our test student is our value for the parameter *k*. In this context, similarity is measured by Euclidian distance between data points. Small values of k will be easy to evaluate, but perhaps less accurate. If we set k = 1, we will only look for the single other student most like our new student. While most students living together are of the same class (athlete vs artist), some students are from the other. This will be quick but will frequently predict the wrong category. In our example, the closest student to our new student is an athlete, so we (incorrectly) predict our new student will also be an athlete.

A screenshot of a cell phone

Description automatically generated

However, if we increase the number of nearby students to four (k=4), we now notice that we now have 3 artists in additional to the athlete, all of whom live in the artist dorm. When the k-nearest neighbors to our test point are of multiple types, we pick the modal type. We now correctly classify the new student as an Artistic dorm student. In this instance, identifying the four closest students will take longer than simply identifying the first. We have traded a shorter processing time for a more accurate model.

A screenshot of a cell phone

Description automatically generated

As we increase \*k\*, we will generally get more accurate results, but each computation will take more and more time. Due to idiosyncrasies in our data, certain values of k will be more accurate than others. Therefore, we ultimately must decide our value of k based on the accuracy of that model produced by that particular value weighed against the computation time for that value.

**Applying KNN to Our NASA Data**

We will now shift from our hypothetical university to a real-world application. The rest of this tutorial will use NASA contract data 2017 combined with state level data from the same time.[[4]](#footnote-4) We will use KNN to attempt to predict whether a given state has one a Senator sitting on the Senate Commerce Subcommittee on Space, Science, and Competitiveness; the committee has since morphed into the Subcommittee on Aviation and Space in the 116th Congress.[[5]](#footnote-5)

Training-Test Split

As with most machine learning models, our first step in applying our model is creating a training-test split in the data. This allows us to train the model on some of the data and test the accuracy of our model on the test data. We'll use the train\_test\_split function in scikitlearn to do this automatically for us. Normally, a training-test split is either 80-20 or 90-10, but for illustrative purposes, I've done a 70-30 split.[[6]](#footnote-6)

Our *feature matrix,* the data frame that contains our independent variables, consists of the following for each state in 2017: the logged population, the infant mortality rate, the number of infant deaths, a dummy variable representing whether the state is in the top ten states with the highest NASA contract award totals (1=yes), and a dummy variable representing whether the state has a NASA facility within it (1=yes). Our *target array* is the list of values for the column indicating whether a state has a Senator on the subcommittee.

A picture containing sky

Description automatically generatedMoving Forward

Once we have fit our model, we can begin to tune it. Since the aim of this tutorial is explaining the logic behind KNN, we won’t go very far into the fine-tuning and validation stages of the modeling process. The most straight-forward way to evaluate our value for k is to plot it against the resulting accuracy score from our KNN model. We see that k = 5 produces the most accurate model. Any useful application of KNN would then continue to validate the model using an ROC curve and confusion matrix, but these are topics for another post.

1. The code to produce the plots and apply knn is available at <https://github.com/DHummel-Price/knntutorial> [↑](#footnote-ref-1)
2. For more information about mapping variables to n-dimensional spaces, see:

   <https://towardsdatascience.com/understanding-high-dimensional-spaces-in-machine-learning-4c5c38930b6a> [↑](#footnote-ref-2)
3. For more information on the logic behind training-test splits, see:

   <https://medium.com/datadriveninvestor/data-science-essentials-why-train-validation-test-data-b7f7d472dc1f> [↑](#footnote-ref-3)
4. This is the file called “Cleaned\_2017\_NASA\_POP\_INFMORT.csv” from the git repo. The data for this next section was wrangled and cleaned in the Jupyter Notebook call "Data\_Cleaning\_and\_Merging". [↑](#footnote-ref-4)
5. <https://www.commerce.senate.gov/aviation-and-space> [↑](#footnote-ref-5)
6. Due to the random nature of this splitting function, some splits will result in more accurate models than others. Particularly when the number of observations is small (arbitrarily: less than 200), the appropriate way to evaluate our model will be with a cross-val score. This is beyond the scope of this tutorial. [↑](#footnote-ref-6)