Lecture 5: Introduction to Supervised Learning

Intro to Data Science for Public Policy, Spring 2016

by Jeff Chen & Dan Hammer, Georgetown University McCourt School of Public Policy

Contents

Supervised learning is used everyday whether in the social and natural sciences or web development or public policy. The term "supervised learning" comes from the nature of the empirical task, where an algorithm is *trained* to replicate labeled examples using input features and uses objective functions to minimize error or maximize information gain. Labeled examples are also known as dependent variables or target variables), and *input features* are also known as independent variables or predictors. This assumes that some combination of features can be used to describe and predict targets. For example:

1. **Public Health**. Classifying whether a restaurant is at risk of violating city health regulations using past ratings as a target and restaurant reviews as an input.
2. **Labor**. Estimating the impact of minimum wage on business health by using employment as a target and changes in labor laws as inputs.
3. **Environment**. Predicting whether a storm detection signature as identified in weather radar will result in property damage by using human reported damage as the target and the physical characteristics of storm cells as inputs.
4. **Housing**. Estimating the economic impact of opening a landfill by using housing prices as a target conditioned upon housing characteristics and distance from the landfill as inputs.
5. **Health**. Estimating one's level of physically activity using smartphone accelerometer data as inputs and user-tagged labels as targets.
6. **Operations**. Forecasting call volumes at a call center to help set appropriate staffing levels to meet citizen demand.

Thinking about supervised learning

The structure of a supervised learning project is much like taking a course on any academic subject. Let's take the example of a calculus class. Students are taught concepts and the application of those concepts through readings and homeworks. Each concept has a structure that is learned by examining and working through practice problems that are representative of that concept. For example, when working with derivatives, the first derivative of $f(x) = x^2$ is $f'(x) = 2x$ and its second derivative is $f''(x) = 2$. Under certain conditions, certain mathematical functions such as exponential and logarithmic functions behave differently, and the rules and patterns for handling those derivatives need to be taken into account when approaching derivative problems. Eventually, the professor will schedule and administer a midterm or final covering all the different learned concepts. In preparation, a series of practice exams are typically provided to students as a way to test their knowledge. A practice exam is most helpful when the material has already been well-studied and is taken only once, using examples that were answered incorrectly as an opportunity to provide insight into gaps in knowledge. Otherwise, repeatedly taking the same practice exam will provide a false sense of mastery as a student takes and re-takes and learns the answers as opposed to the eccentricities of the subject at hand. The real test is the actual exam, in theory indicating how well concepts were understood and re-applied.

There are parallels between taking a class and supervised learning tasks. In a classroom setting, homeworks and practice tests are used to build applied skills to accomplish specific tasks, whereas the actual exam is used to determine precisely how robust those skills are. In supervised learning, the data is often times partitioned into two or more parts, then an algorithm is *trained* on at least one partition to learn underlying patterns, then the learned rules and weights are applied to the remaining part of the *hold out* sample to *test* the accuracy of the algorithm.

In application, we can approach a supervised learning problem by how one intends to use the outputs of the algorithm and the design of the algorithmic experiment.

Supervised Learning as a Process

Algorithms are often times mentioned in technical discussions. *Artificial neural networks* and *support vector machines* are commonly found in computer vision tasks. *Linear regression* for social science, natural science and and general use. *K-Nearest Neighbors* and *K-means* clustering are common in retail and e-commerce. But ultimately, the algorithm is only one part of a whole process. Applied supervised learning is a process that is guided by a number of core considerations, including core considerations: (1) Intended use, (2) Data, (2) Experiment design, (4) Algorithm development, (5) Deployment

(1) Intended use

The intended use influences the type of algorithm. It all starts with a well-formulated data-enable question that can be answered using an empirical outcome. For example:

1. "Which of the current lawsuits in the legal department's backlog can be won?"
2. "Which prospective patients will require advanced medical treatment in the next 90 days?"
3. "What are the greatest drivers of lawsuit generation over the last 10 years?"
4. "Which patient characteristics are most associated with stroke?"

In each of these questions, a quantitative value can be used to answer a question; however, there are nuances. The first two questions are focused on *prediction* – often times a *score* that represents the chances of a outcome. The goal is to create a sure-fire, highly accurate function that can be relied upon to make solid decisions without much manual effort, but without a strong emphasis on interpreting underlying relationships. For example, prediction projects can be used to prioritize risky buildings for inspection, screen loan applicants for financial default, and surface search results. Typically, prediction problems are required for problems of significant scale, where it becomes cost prohibitive for humans to do a task themselves. A modern example is the application of computer vision to score whether certain features are contained in images. In 2012, a Google demonstration project using artificial neural networks showed how a computer vision algorithm can detect whether a cat is present in hundreds of thousands photographs with a high degree of accuracy. While the algorithm is accurate, the underlying "machinery" was not designed to be easily interpretted as prediction projects. Prediction projects are more operations or action-focused, often times designed to be put into *production*, or implemented to support everyday tasks.

The latter two questions are more concerned with *estimating relationships* to understand if one or more inputs are empirically associated with a target variable. While the methodologies used for estimation can yield accurate predictions, the goalis to calibrate coefficients and weights in a model that shows how an input feature is associated or potentially impacts the target. For example, a highway traffic study may find that a 1% increase in employment is associated with a 0.5% increase in highway traffic volume. July 4th weekend may increase emergency call volume by a certain percentage. Estimation often relies on linear regression methods, which are covered later in this chapter.

In either case, it is helpful to think about supervised learning as a pointer. It is easy to flag potential cases of an observed phenomena (e.g. what, which, who, where, when), but it is hard to use supervised learning to definitively provide an answer as to "why" something happens.

(2) The Data

Upon formulating research questions, the next priority is finding usable and actionable data. At a minimum, data need to contain:

- *A target variable or labels.* Each record should be furnished with labels of what needs to be predicted. Labels can take on any structured form such as discrete or continuous values.

- *Input features.* A series features that be used to relate and predict the target.

In addition, the qualities of the data must be aligned with the task at hand:

- *Unit of Analysis.* A well-formulated research question has a clear unit of analysis. The data must reflect the requirements of the question. A driverless car, for example, likely requires updated telemetry data in sub-second frequencies in order to accurately and safely maneuver. Providing telemetry data every five minutes do not likely meet the requirements.
- *Data Pipeline.* If the data-enabled question will be repeatedly asked, it is worth examining the reliability of the *data pipeline* and whether new data will be available when the question is asked in the future. For example, more strategic problems like a 10-year population forecast may only be conducted every year requiring data once a year, whereas more operational problems like restaurant inspections may be done on a daily basis requiring new data in order to detect new patterns every day.

(3) Experiment Design

Cross Validation. Supervised learning problems usually involve partitioning data to help with optimizing algorithm for accuracy and reduce the chance of *overfitting*, a condition in which an algorithm learns patterns that are noise as opposed to signal thereby leading to misleading inferences. Partitioning involves splitting the data into two or more sets where one method is "held out" from training algorithms and the other set is used for validating and tuning results. Data scientists commonly rely on two partition procedures: - *Train/Validate/Test* assumes that the data are partitioned into three sets. The *train* set is used to initially calibrate the algorithm. The *validation* set is used to help tune hyperparameters and select features. Typically, the algorithm that is calibrated in the training stage is used to predict values in the validation set and as this set contains labels, accuracy can be assessed using appropriate measures such as RMSE or AUC. Once it is determined the algorithm is as good as it can be, the trained algorithm is then used to score a set of remaining examples in the *test* set in order to assess its generalizability. These three samples may be partitioned in the following proportions: train = 70%, validate = 15%, and test = 15%. 70/15/15 for short. - *K-Folds Cross Validation.* A train/validate/test design can be extended for more exhaustive model tuning. K-folds cross validation involves partitioning the data into k partitions. Then, combine $k-1$ partitions to train an algorithm and predict the values for the k^{th} part. Then, cycle through combinations of $k-1$ partitions until each of the k holdout samples have been predicted. Upon doing so, the prediction accuracy can be calculated for each of the k partitions as well as for all k partitions together. Partitions that yield poorer accuracy relative to other partitions help provide a clue as to when an algorithm is insufficient or requires further tuning. For exhaustive testing, $k = n-1$ such that $n-1$ models are trained such that each of the n records in a data set are predicted once.

For data where no clear labels are available, we may rely on "unsupervised learning" – a class of tasks that used to find patterns, structure, and membership using input features alone. Unsupervised learning encompasses clustering (e.g. values may naturally fall into clusters in n-dimensional space) and dimensionality reduction. We will cover unsupervised learning in Lecture 9.

(4) Algorithm

3. *A solvable algorithm or technique.* Techniques are concerned with the treatment of the type of labeled data, which has particular influence on the structure and assumptions of mathematical operations. In supervised learning, there are two broad categories of algorithms, including:

- *Regression.* A number of the of the examples may depend on *regression*, which is a statistical method for estimating the relationship between a set of features or variables. Regression problems are formulated with a dependent variable that is trained or conditioned upon independent variables with the goal of estimating the expected value of given a set of variables. of calibrating *coefficients* or *weights*, which are used to not only produce a prediction of the dependent variable but infer the contribution of an independent variable if which is a class of algorithms that estimate the expected value of a target conditioned upon (e.g. examples #2 and #4). Common examples of regression models include Ordinary Least Squares (covered in this chapter) and Logistic Regression (covered in Chapter 8).

- *Classifiers.* The remaining are *classifiers*, or algorithms designed to predict membership to discrete categories.
  [hyperparameters]

In addition, algorithms typically are evaluated on [x]

k-Nearest Neighbors (kNN)

k-nearest neighbors (KNN) is a non-parametric pattern recognition algorithm that is based on a simple idea: observations that are more similar will likely also be located in the same neighborhood. Given a class label y associated with input features x, a given record i in a dataset can be related to all other records using Euclidean distances in terms of x:

$$\text{distance} = \sqrt{\sum (x_{ij} - x_{ij})^2}$$

where j is an index of features in x and i is an index of records (observations). For each i, a neighborhood of k records can be determined using the ranked ascending distance to all other records. The value of y for record i can be approximated by the k neighbors that surround i. For discrete target variables, y_i is determined using a procedure called *majority voting* where the most prevalent value in the neighborhood around i is assigned. For continuous variables, the neighborhood mean is used to approximate y_i.

How does one implement this exactly? To show this process, pseudocode will be relied upon. It's an informal language to articulate and plan the steps of an algorithm or program, principally using words and text as opposed to formulae. There are different styles of pseudocode, but the general rules are simple: indentation is used to denote a dependency (e.g. control structures). For all techniques, we will provide pseudocode, starting with KNN:

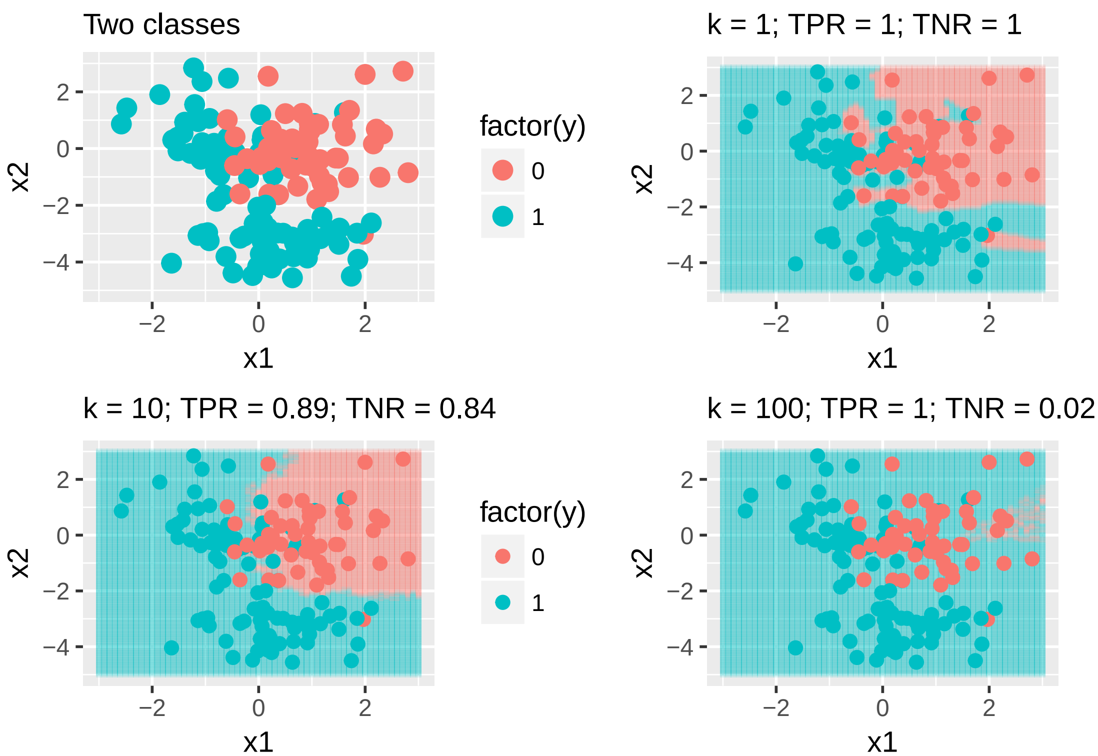
Pseudocode

```
kNN( k, set, y, x){
  Pre-Process (optional):
    > Transform or standardize all input features

  Loop through each `item` in `set`{
    > Calculate vector of distances in terms of x from `item` to all other items in `set`
    > Rank distance in ascending order

    if target `y` is continuous:
      > Calculate mean of `y` for items ranked 1 through k
    else if target is discrete:
      > Calculate share of each discrete level for items ranked 1 through k
      > Use majority voting to derive expected value
  }
```

}

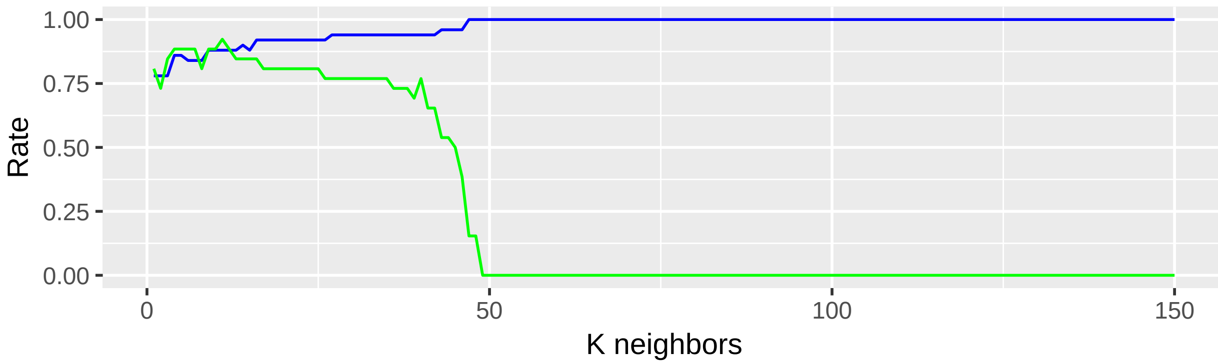
The procedure described above yields the results for just one value of k. However, kNNs, like many other algorithms, are an iterative procedure, requiring tuning of *hyperparameters* – or values that are starting and guiding assumptions of a model. In the case of kNNs, k is a hyperparameter and we do not precisely know the best value of k. Often times, tuning of hyperparameters involve a *grid search*, which is a process that involves systematic testing of along equal intervals of the hyperparameter.

Below, a grid search has been conducted for a kNN along intervals of a $log_1 0$ scale. The large points represent a training set and the surrounding area has been *scored* or predicted to show the shape of the region corresponding each value of k.



Which K is the right K?

The accuracy of a KNN model is principally dependent on finding the right value of k directly determines what enters the calculation used to predict the target variable. Thus, to optimize for accuracy, try multiple values of k and compare the resulting accuracy values. It is helpful to first see that when $k = n$, kNNs are simply the sample statistic (e.g. mean or mode) for the whole dataset. Below, the True Positive Rate (TPR, blue) and True Negative Rate (TNR, green) have been plotted for values of k from 1 to n. The objective is to ensure that there is a balance between TPR and TNR such that predictions are accurate. Where $k > 20$, the TPR is near perfect. For values of $k < 10$, TPR and TNR are more balanced, thereby yielding more reliable and accurate results.



There are other factors that influence the selection of k:

- Scale. kNNs are strongly influenced by the scale and unit of values of x as ranks are dependent on straight Euclidean distances. For example, if a dataset contained random measurements of age in years (a relatively nor) and wealth in dollars, the units will over emphasize income as the range varies from 0 to billions whereas age is on a range of 0 to 100+. To ensure equal weights, it is common to transform variables into standardized scales such as:
  - Range scaled or
  $$\frac{x - \min(x)}{\max(x) - \min(x)}$$
  yields scaled units between 0 and 1, where 1 is the maximum value
  - Mean-centered or
  $$\frac{x - \mu}{\sigma}$$
  yield units that are in terms of standard deviations
- Symmetry. It's key to remember that neighbors around each point will not likely be uniformly distributed. While kNN does not have any probabilistic assumptions, the position and distance of neighboring points may have a skewing effect.

Usage

KNNs are efficient and effective under certain conditions. First, kNNs can handle target values that are either discrete or continuous, making the approach relatively flexible. They are best used when there are relatively few features as distances to neighbors need to be calculated for each and every record and need to be optimized by searching for the value of `k` that optimizes for accuracy. In cases where data is randomly or uniformly distributed in fewer dimensions, a trained KNN is an effective solution to filling gaps in data, especially in spatial data. However, kNNs are not interpretable as it is a nonparametric approach – it does not produce results that have a causal relationship or illustrate. Furthermore, kNNs are not well-equipped to handle missing values.

An Example

In practice in `R`, KNNs can be trained using the `knn()` function in the `class` library. However, this function is best suited for discrete target variables. To illustrate KNN regressions, we will write a function from scratch and illustrate using remote sensed data. Remote sensing is data obtained through scanning the Earth from aircrafts or satellites. Remote sensed earth observations yield information about weather, oceans, atmospheric composition, human development among other things – all are fundamental for understanding the environment. As of Jan 2017, the National Aeronautics and Atmospheric Administration (NASA) maintains two low-earth orbiting (LEO) satellites named Terra and Aqua, each of which takes images of the Earth using the Moderate Resolution Imaging Spectroradiometer (MODIS) instrument. Among the many practical scientific applications of MODIS imagery is the ability to sense vegetation growth patterns using the Normalized Difference Vegetation Index (NDVI) – a measure ranging from -1 to +1 that indicates that amount of live green on the Earth's surface. Imagery data is a $n \times m$ gridded matrix where each cell represents the NDVI value for a given latitude-longitude pair.

NASA's Goddard Space Flight Center (GSFC) publishes monthly MODIS NDVI composites. For ease of use, the data has been reprocessed such that data are represented as three columns: latitude, longitude, and NDVI. In this example, we randomly select a proportion of the data (~30%), then use KNNs to interpolate the remaining 70% to see how close we can get to replicating the original dataset. In application, scientific data that is collected *in situ* on the Earth's surface may take on a similar format – represembling randomly selected points that can be used to generalize the measures on a grid, even where measures were not taken. This process of interpolation and gridding of point data is the basis for inferring natural and manmind phenomena beyond where data was sampled, whether relating to the atmosphee, environment, infrastructure, among other domains.

To start, we'll set a working directory.

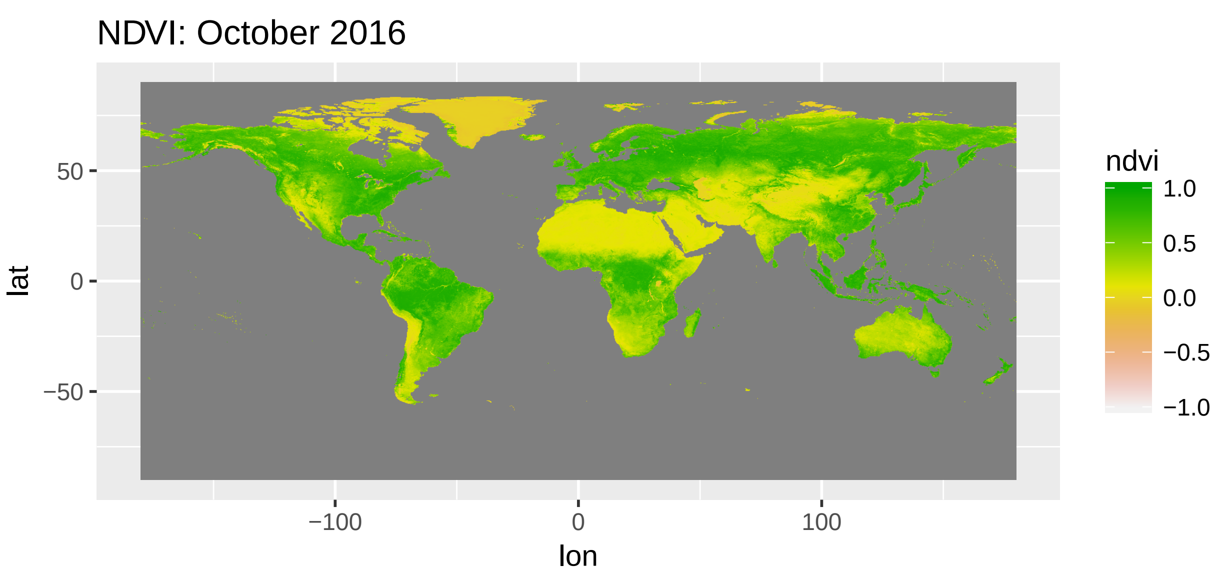
```
setwd("[your-dir]")
```

Then read in the data, which is available in CSV form on the course repo.

```
#Read CSV of data
df <- read.csv("ndvi_sample_201606.csv")
```

To view the data, we can use the `geom_raster()` option in the `ggplot2` library. Notice the color gradations between arrid and lush areas of vegetation.

```
library(ggplot2)
ggplot(df, aes(x=lon, y=lat)) +
          geom_raster(aes(fill = ndvi)) +
          ggtitle("NDVI: October 2016") +
          scale_fill_gradientn(limits = c(-1,1), colours = rev(terrain.colors(10)))
```



The NDVI data does not provide values on water. As can be seen below, cells that do not contain data are represented as 99999 and are otherwise values between -1 and +1.

```
##       lat      lon         ndvi
## 1  89.875 -179.875 9.999900e+04
## 2  89.625 -179.875 9.999900e+04
## 3  89.375 -179.875 9.999900e+04
## 74 71.625 -179.875 1.047244e-01
## 75 71.375 -179.875 4.472441e-01
```

```
## 76 71.125 -179.875 3.763779e-01
```

For this example, we will focus on an area in the Western US and extract only a 30% sample.

```
#Subset image to Western US near the Rocky Mountains
  us_west <- df[df$lat < 45 & df$lat > 35 &  df$lon > -119 & df$lon < -107,]

#Randomly selection a 30% sample
  set.seed(32)
  sampled <- us_west[runif(nrow(us_west)) < 0.3 & us_west$ndvi != 99999,]
```

A KNN algorithm is fairly simple to build when the scoring or voting function is a simple mean. All that is required is to write a series of a loops to calculate the nearest neighbors for any value of `k`. The `knn.mean` function should take a training set (input features - `x_train` and target - `y_train`), and a test set (input features - `x_test`).

```
knn.mean <- function(x_train, y_train, x_test, k){

  #Set vector of length of test set
    output <-  vector(length = nrow(x_test))

  #Loop through each row of the test set
    for(i in 1:nrow(x_test)){

      #extract coords for the ith row
        cent <- x_test[i,]

      #Set vector length
        dist <- vector(length = nrow(x_train))

      #Calculate distance by looping through inputs
        for(j in 1:ncol(x_train)){
          dist <- dist + (x_train[, j] - cent[j])^2
        }
        dist <- sqrt(dist)

      #Calculate rank on ascending distance, sort by rank
        df <- data.frame(id = 1:nrow(x_train),rank = rank(dist))
        df <- df[order(df$rank),]

      #Calculate mean of obs in positions 1:k, store as i-th value in output
        output[i] <- mean(y_train[df[1:k,1]], na.rm=T)
    }
  return(output)
}
```

The hyperparameter `k` needs to be tuned. We thus also should write a function to find the optimal value of `k` that minimizes the loss function, which is the Root Mean Squared Error (RMSE $= \sigma = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}.}$).

```
knn.opt <- function(x_train, y_train, x_test, y_test, max, step){

  #create log placehodler
  log <- data.frame()

  for(i in seq(1, max, step)){
    #Run KNN for value i
```

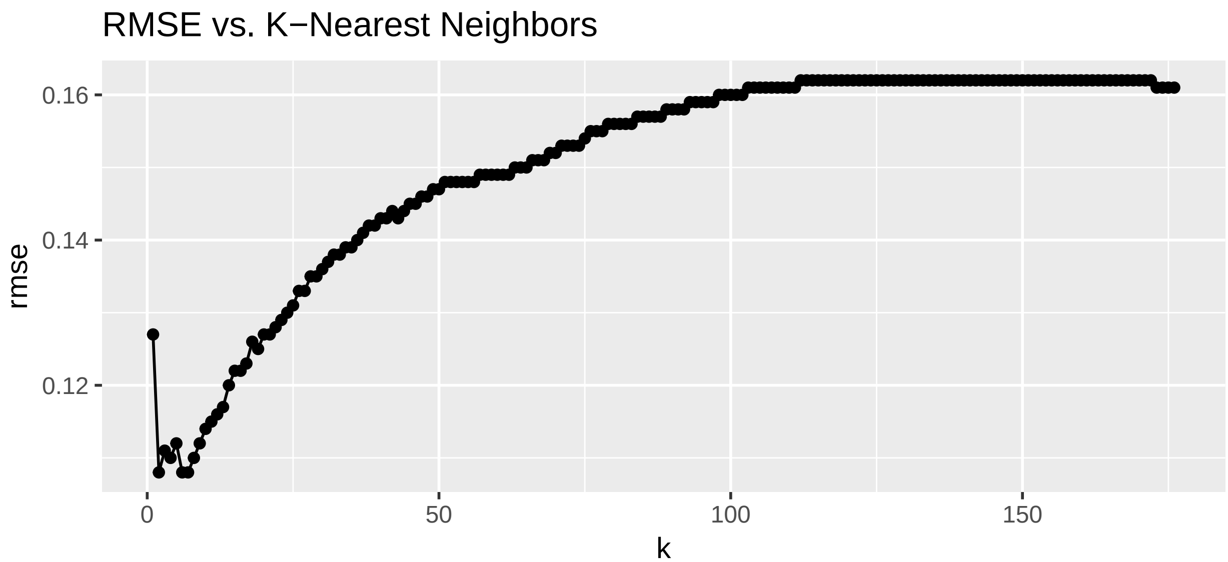
```
      yhat <- knn.mean(x_train, y_train, x_test, i)

    #Calculate RMSE
      rmse <- round(sqrt(mean((yhat  - y_test)^2, na.rm=T)), 3)

    #Add result to log
      log <- rbind(log, data.frame(k = i, rmse = rmse))
  }

  #sort log
  log <- log[order(log$rmse),]

  #return log
  return(log)
}
```

Normally, the input features (e.g. latitude and longitude) should be normalized, but as the data are in the same coordinate system and scale, no additional manipulation is required. From the 30% sampled data, a training set is subsetted containing 70% of sampled records and the remaining is reserved for testing.

```
#Set up data
  set.seed(123)
  rand <- runif(nrow(sampled))

#training set
  xtrain <- as.matrix(sampled[rand < 0.7, c(1,2)])
  ytrain <- sampled[rand < 0.7, 3]

#test set
  xtest <- as.matrix(sampled[rand >= 0.7, c(1,2)])
  ytest <- sampled[rand >= 0.7, 3]
```

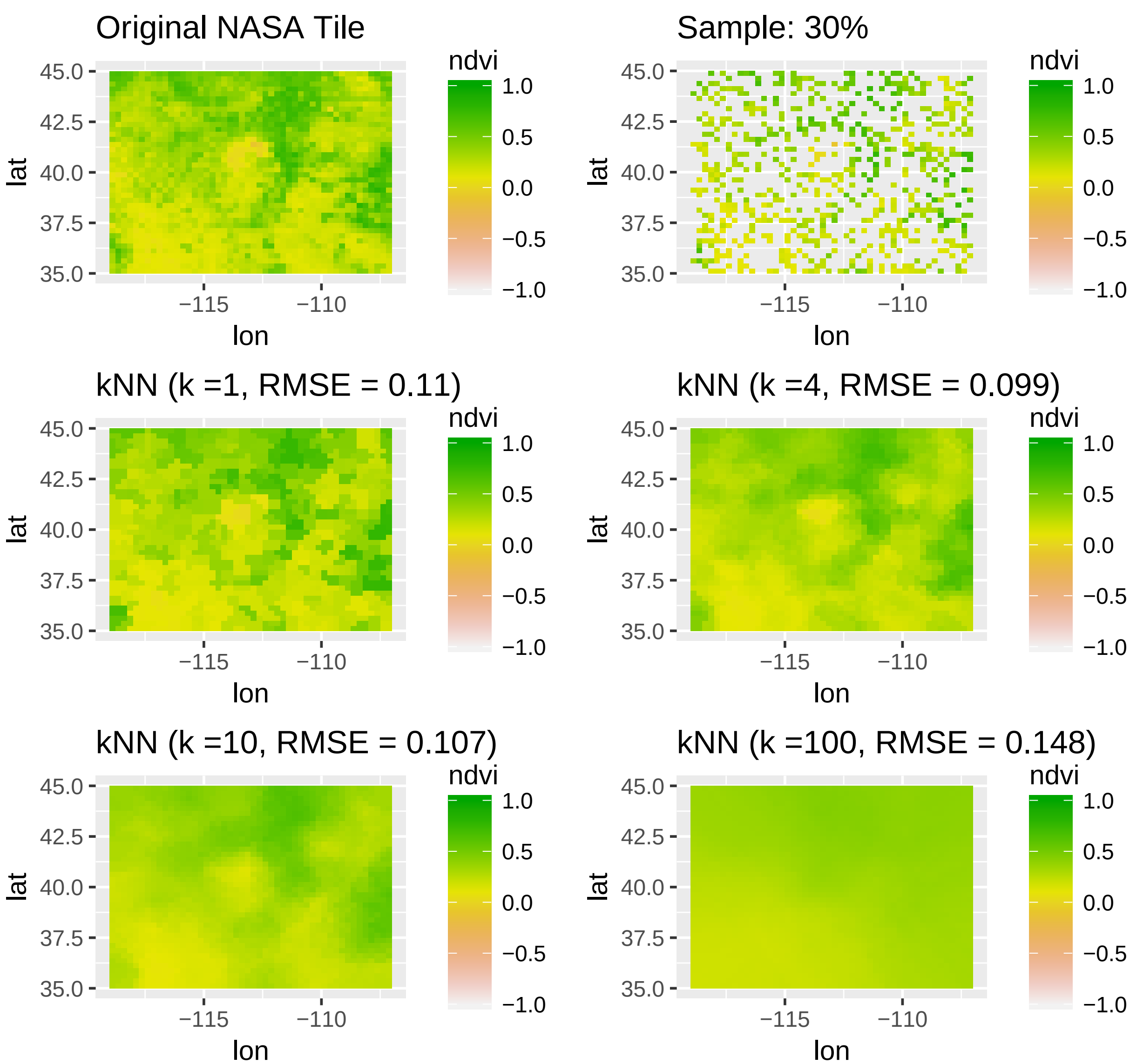
The algorithm can now be placed into testing, searching for the optimal value of `k` along at increments of `1` from $k = 1$ to \$ k = n\$. Based on the grid search, the optimal value is $k = 4$.

```
#opt
  logs <- knn.opt(xtrain, ytrain, xtest, ytest, nrow(xtest), 1)

#Plot results
  ggplot(logs, aes(x = k, y = rmse)) +
          geom_line() + geom_point() + ggtitle("RMSE vs. K-Nearest Neighbors")
```



With this value, we can now put this finding to the test by plotting the interpolated data as a raster. Using the `ggplot` library, we will produce six graphs to illustrate the tolerances of the methods: the original and sampled images as well as a sampling of rasters for various values of k.

```
#Original
  full <- ggplot(us_west, aes(x=lon, y=lat)) +
            geom_raster(aes(fill = ndvi)) +
            ggtitle("Original NASA Tile") +
            scale_fill_gradientn(limits = c(-1,1), colours = rev(terrain.colors(10)))

#30% sample
  sampled <- ggplot(sampled, aes(x=lon, y=lat)) +
            geom_raster(aes(fill = ndvi)) +
            ggtitle("Sample: 30%") +
            scale_fill_gradientn(limits = c(-1,1), colours = rev(terrain.colors(10)))

#Set new test set
  xtest <- as.matrix(us_west[, c(1,2)])

#Test k for four different values
  for(k in c(1, 4, 10, 100)){
    yhat <- knn.mean(xtrain,ytrain,xtest, k)
    pred <- data.frame(xtest, ndvi = yhat)
    rmse <- round(sqrt(mean((yhat  - us_west$ndvi)^2, na.rm=T)), 3)

    g <- ggplot(pred, aes(x=lon, y=lat)) +
      geom_raster(aes(fill = ndvi)) +
      ggtitle(paste0("kNN (k =",k,", RMSE = ", rmse,")")) +
      scale_fill_gradientn(limits = c(-1,1), colours = rev(terrain.colors(10)))

    assign(paste0("k",k), g)
  }

  #Graphs plotted
    library(gridExtra)
    grid.arrange(full, sampled, k1, k4, k10, k100, ncol=2)
```



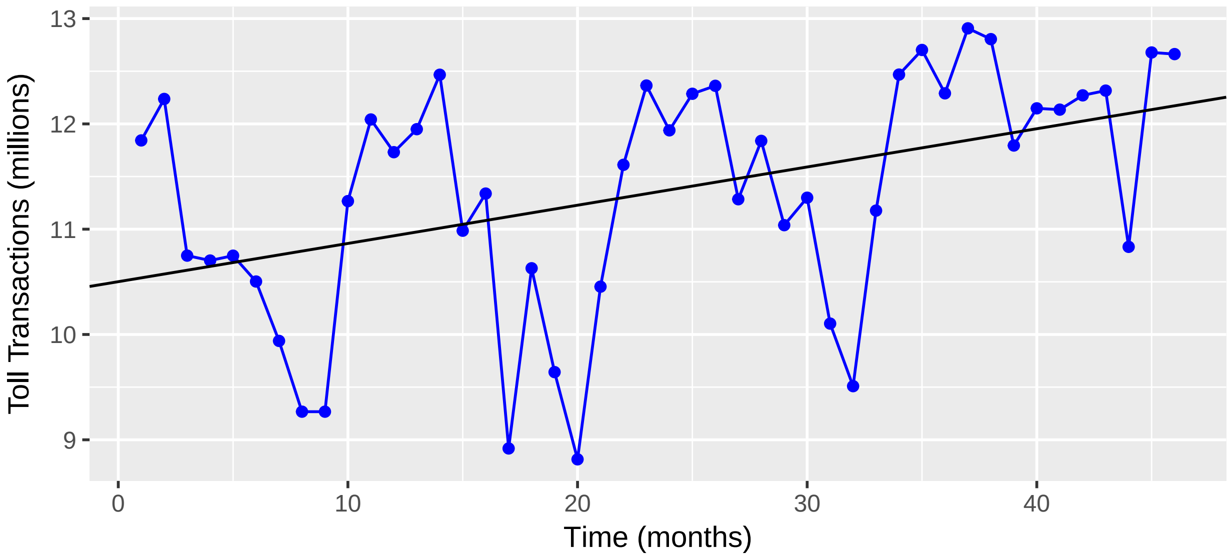
Exercises 5.1

1. For the following values, write a function to retrieve the value of y where $k = 1$ for each i.
2. Modify the function to handle k = 2.

Ordinary Least Squares (OLS) Regression

Every year, cities and states across the United States publish measures on the performance and effectiveness of operations and policies. Performance management practitioners typically would like to know the direction and magnitude, as illustrated by a linear trend line. Is crime up? How are medical emergency response times? Are we still on budget? Which voting blocks are drifting?

For example, the monthly number of toll transactions in the State of Maryland is plotted over time from 2012 to early 2016. The amount is growing with a degree of seasonality. But to concisely summarize the prevailing direction of toll transactions, we can use a trend line. That trend line is an elegant solution that shows the shape and direction of a linear relationship, taking into account all values of the vertical and horizontal axes to find a line that weaves through and divides point in a symmetric fashion.



This trend line can be simply described in using the following formula:

$$\text{transactions} = 10.501 + 0.036 \times \text{months}$$

and every point plays a role. We can infer that the trend grows at approximately 36,000 transactions per month. Using the observed response y and the independent variable x, calculating the intercept and slope is a fairly simple task:

$$\text{slope} = \hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

and

$$\text{intercept} = \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

In a bivariate case such as this one, it's easy to see the interplay. In the slope, the covariance of X and Y ($\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$) is tempered by the variance of x ($\sum_{i=1}^{n}(x_i - \bar{x})^2$). If the covariance is greater than the variance, then the absolute value of the slope will be greater than one. The direction of the slope (positive or negative) is determined by the interaction between x and y alone.

Trend lines are one of many uses of a class of supervised learning called regression, but best known of which is Ordinary Least Squares or Least Squares Regression. In multivariate cases where many variables are used to get a handle on what factors influence y, the problem gets more complex. Nonetheless, OLS regression is the quantitative workhorse of data-driven public policy. The technique is a statistical method that estimates unknown parameters by minimizing the sum of squared differences between the observed values and predicted values of the target variable.

To better understand arguably the most commonly used supervised learning method, we can start by defining a regression formula:

$$y_i = w_0 x_{i,0} + w_1 x_{i,1} + ... + w_k x_{i,k} + \epsilon_i$$

where:

- y_i is the target variable or "observed response"
- w_k are coefficients associated with each x_k. Note that w may be substituted with β in some cases.
- $x_{i,k}$ are input or independent variables

- subscript i indicates the index of individual observations in the data set
- k is an index of position of a variable in a matrix of x
- ϵ_i is an error term that is assumed to have a normal distribution of $\mu = 0$ and constant variance σ^2

Note that $x_{i,0} = 1$, thus w_0 is often times represented on its own. For parsimony, this formula can be rewritten in matrix notation as follows:

$$y = w^T x$$

such that y is a vector of dimensions $n \times 1$, x is a matrix with dimensions $n \times k$, and w is a vector of length k.

Given this formula, the objective is to minimize the Sum of Squared Errors as defined as

$$SSE = \sum_{i=0}^{n} (y_i - \hat{y})^2$$

or the Mean Squared Error as defined as

$$MSE = \frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2$$

. Both measures require the *predicted value* of y as calculated as

$$\hat{y}_i = w_0 + w_1 x_{i,1} + ... + w_k x_{i,k}$$

. The SSE and MSE are measures of uncertainty relative to the observed response. Minimization of least squares can be achieved through a method known as *gradient descent*.

[More on gradient descent here]

Assumptions

Interpretation

There are a number of attributes and outputs of a linear squares regression model that are examined, namely the R-squared, coefficients, and error. *R-squared* or R^2 is a measure of the proportion of variance of the target variable that can be explained by a estimated regression equation. A few key bits of information are required to calculate the R^2, namely:

- $\bar{y}$: the sample mean of y;
- $\hat{y}_i$: the predicted value of y for each observation i as produced by the regression equation; and
- y_i: the observed value of y for each observation i.

Putting these values together is fairly simple:

- Total Sum of Squares or TSS is the variance of y:

$$\text{TSS} = \sigma^2(y) = \sum_{i=1}^{n}(y_i - \hat{y})^2$$

- Sum of Squared Errors is the squared difference between each observed value of y and its predicted value $\hat{y}_i$:

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Regression Sum of Squares or RSS is the difference between each predicted value $\hat{y}_i$ and the sample mean $\bar{y}$.

Bringing all the values together,

$$R^2 = 1 - \frac{SSE}{TSS}$$

. As TSS will always be the largest value, R^2 will always be bound between 0 and 1 where a value of $R^2 = 0$ indicates a regression line in which x does not account for variation in the target whereas $R^2 = 1$ indicates a perfect regression model where x accounts for all variation in y.

In addition, Root Mean Square Error (RMSE) is helpful for understanding the variation of the predictions relative to y. RMSE is defined as

$$\text{RMSE} = \sigma = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

. Note that RMSE is interpreted in terms of levels of y, which may not necessarily facilitate easy communication of model accuracy. In certain scenarios, particularly for time series forecasts, Mean Absolute Percentage Error (MAPE) is used to contextualize prediction accuracy relative to y. This measure is defined as

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}|\frac{\hat{y}_i - y_i}{y_i}|$$

.

Under the hood

```
OLS( k, set){

  Define cost function to computer total squared error
  Gradient Descent

}
```