DRAFT - 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

Approaching Supervised Learning

Supervised learning is relied upon everyday, from the social and natural sciences to web development to field operations. Given labeled examples (also known as dependent variables or target variables), a supervised learning task involves *training* a function to mathematically weigh each *feature* in an input set (also known as independent variables or predictors) to replicate the labels (also known as dependent variable or target) for each record in the data. The label can be a continuous variable (e.g. dollar amounts, amount of geographic space, test scores, etc.) or a discrete variable (e.g. yes/no, up/down, walk/jog/run, low/medium/high). The term "supervised learning" comes from the type of task where an algorithm is calibrated based clear, labeled examples and uses objective functions to minimize error or maximize information gain.

A common examples include:

1. **Public Health**. Classifying whether a restaurant is at risk of violating city health regulations using restaurant reviews as an input.
2. **Labor**. Estimating the impact of minimum wage on business health.
3. **Environment**. Predicting whether a storm detection signature as identified in weather radar will result in property damage.
4. **Housing**. Estimating the sales price of houses on the real estate market in order to estimate the impact of opening a landfill.
5. **Health**. Estimating the proportion of one's day that is physically active using smartphone accelerometer data.
6. **Operations**. Forecasting call volumes at a call center to help set appropriate staffing levels to meet citizen demand.

Supervised learning 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 in application. 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.

Extending this basic framework, we can break a supervised learning problem into design considerations and algorithmic considerations.

1. *An intended application.* Applications are concerned with what one would like to infer or do with the data. A well-formulated data-enable question is required, one that is framed and quantified by concrete, well-defined outcome that can be predicted based on quantitative information. For example, "Which of the lawsuits in the backlog are most likely to be lost?", "Which prospective patients will require advanced medical treatment in the next year?", "How many ambulances will need to be available to address medical incidents over the next week?". How a question is formulated is dependent on the intended use of the data, which can split into inferential and prediction tasks:
   - *Estimation and inference.* The former focuses on quantifying relationships in terms of coefficients or weights (e.g. a 1% increase in employment is associated with a 0.5% increase in highway traffic volume), often times relying on linear regression methods.

   - *Prediction.* The latter is focused on maximizing reliability and accuracy of a prediction as opposed to understanding the precise contribution of individual input features. The goal is to create a sure-fire, highly accurate function that can be relied upon to make solid decisions.

2. *A labeled dataset furnished with input features.* 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.
   - *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. *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]

4. *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 - Model and Evaluate

- Target variables
- Input variables
- Objective function and evaluation measures
- AUC
- RMSE
-

In this section, we'll start with [a simple weak]

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

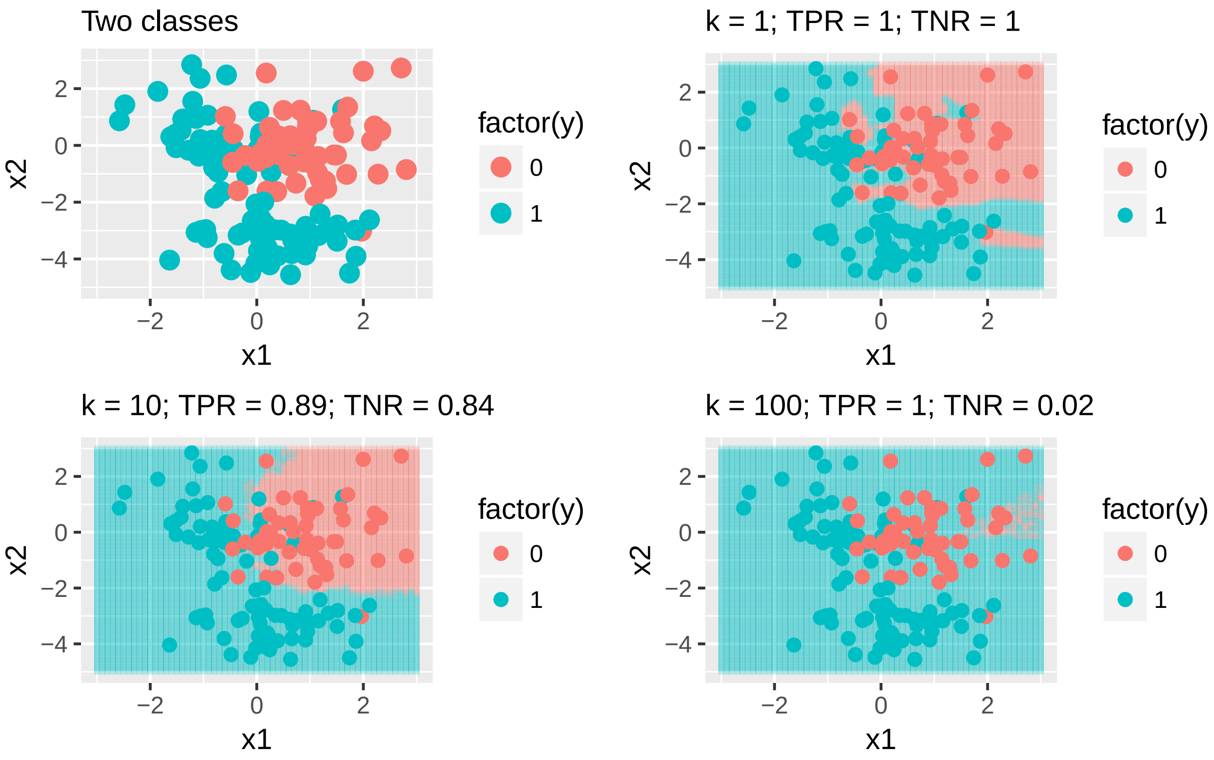
```
kNN( k, set, y, x){
  Pre-Process (optional):
    > Transform or standardize all input features

  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
  }
}
```

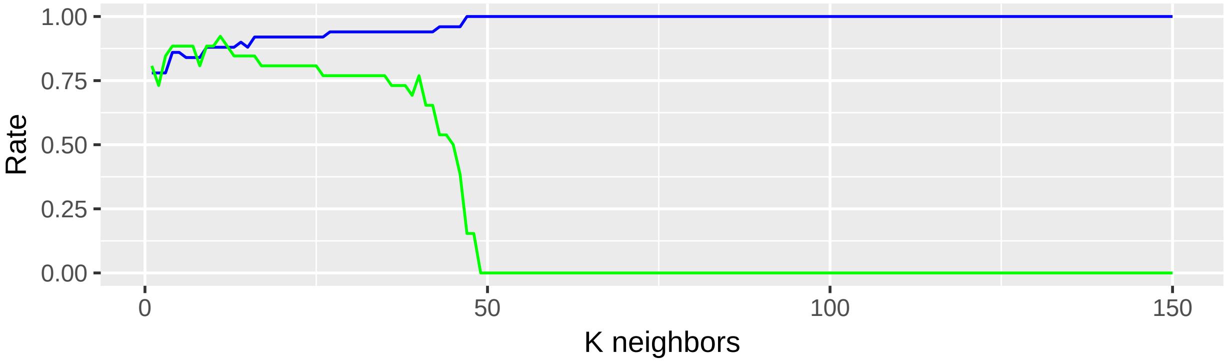
The procedure described above yields the results for just one value of k. However, kNNs, like many other algorithms, are an iterative procedure, requiring tuning of *hyperparameters* – or values that are starting and guiding assumptions of a model. In the case of kNNs, k is a hyperparameter and we do 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_10 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. This

In Practice

```
#KNN code example
```

When to [Not] Use

KNN is - generally slow - not interpretable due to non-parametric approach - Missing values not well handled

- missingness
- variable importance is not considered

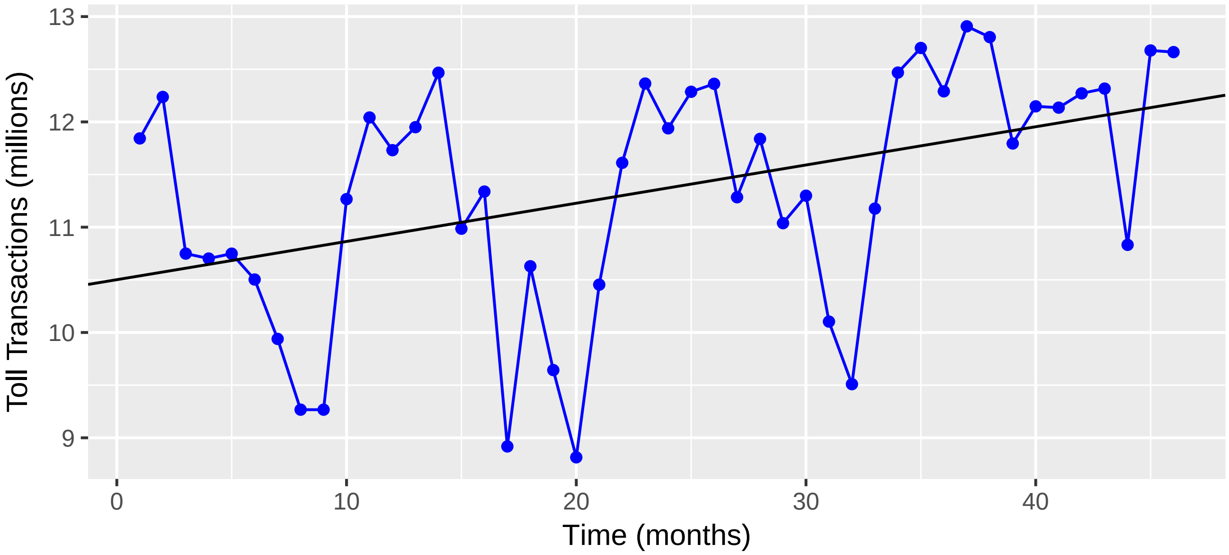
Exercises 5.1

1. For the following values, write a function to retrieve the value of y where $k = 1$ for each i. Then, calculate the True Positive Rate, which is defined as $
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

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

. 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

}
```