Introduction to supervised learning, training vs. validation, and *k*-nearest neighbors

Ryan Miller

Grinnell College
Statistics

Overview

- Applications of *unsupervised learning* to tend to be open-ended and prone to all sorts of subjective choices
  - Methods like scree plots of silhouette scores provide some objective guidance, but it's difficult to judge one analysis to be quantitatively better than another

Overview

- Applications of *unsupervised learning* to tend to be open-ended and prone to all sorts of subjective choices
    - Methods like scree plots of silhouette scores provide some objective guidance, but it's difficult to judge one analysis to be quantitatively better than another
- In contrast, *supervised learning* tends to be highly objective
    - We have a predetermined outcome that we're aiming to predict, which allows us to quantify how accurate our predictions are
- We'll focus on two types of supervised learning
    - **Classification** - predicting the class or category of a data-point
    - **Regression** - predicting a numerical characteristic of a data-point

Supervised learning framework

- Consider data consisting of an n-dimensional vector of outcomes, $\mathbf{y}$, an n by p matrix of features, $\mathbf{X}$
  - Further, suppose the true relationship between $\mathbf{y}$ and $\mathbf{X}$ is given by the following equation:

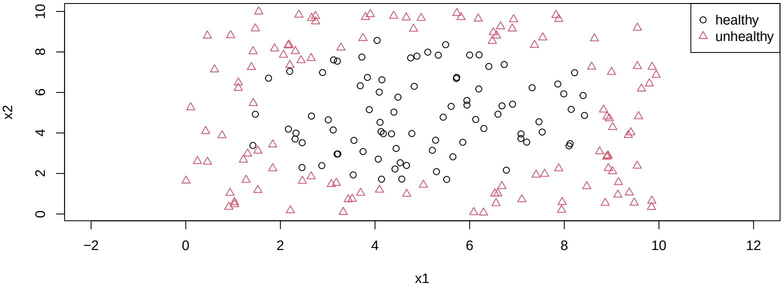
$$\mathbf{y} = f(\mathbf{X}) + \epsilon$$

- The function $f()$ determines how the features in $\mathbf{X}$ influence $\mathbf{y}$
  - ϵ is an n-dimensional vector of errors
- In this setting, we aim to accurately approximate $f()$
  - If $\epsilon = \mathbf{0}$, we may be able to perfectly approximately $f()$
  - However, $\epsilon \neq \mathbf{0}$ introduces *irreducible error*

Grinnell College
Statistics

Example

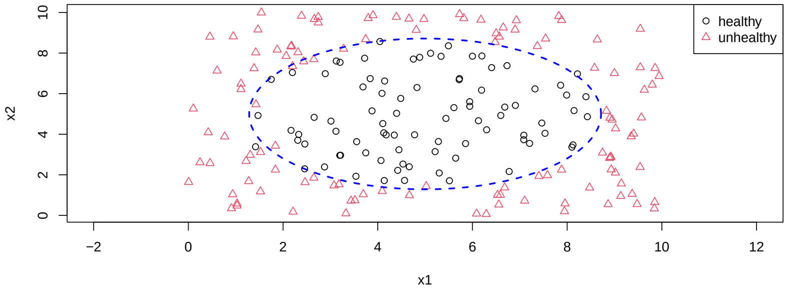
Consider two predictors, X_1 and Y_2, and a binary outcome Y of "healthy" or "unhealthy". Can these predictors be used to accurately *classify* an observation?



Grinnell College
Statistics

Example (cont)

In this example, $f()$ is shown below (blue ellipse):

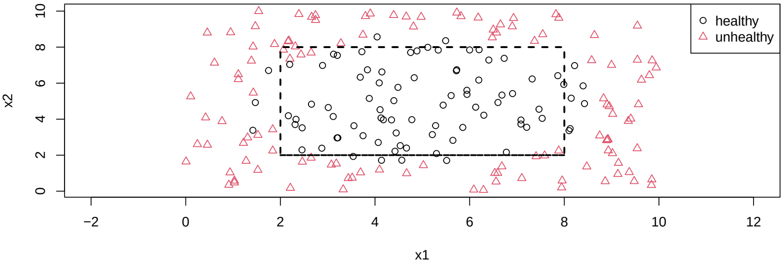


We can see that it's possible to use the data to learn a good approximation of $f()$.

Grinnell College
Statistics

Example (cont.)

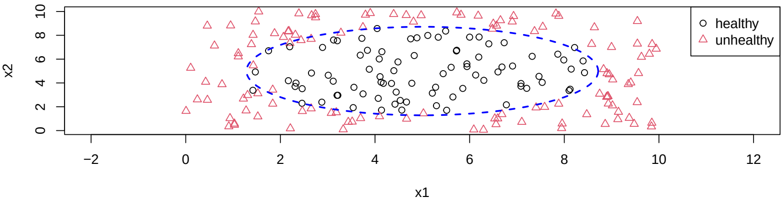
As a human, you might observe that healthy data-points tend to fall between 2 and 8 in both x_1 and x_2, so you might propose the following *classification model*:



This simple model correctly classifies 178 of 200 data-points.

Example (irreducible error)

Let's revisit the true relationship between X_1, X_2, and Y. Notice how some "healthy" data-
points are outside the ellipse, and some "unhealthy" ones are inside it:

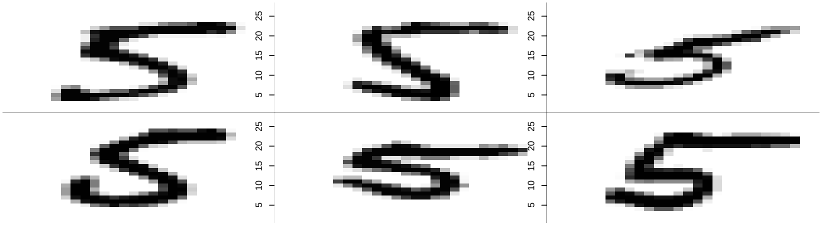


- The misclassification of these data-points reflects this scenario's **irreducible error** (sometimes called "Bayes error")
  - Even the *best possible approximation* of $f()$ cannot perfectly classify every data-point

Grinnell College Statistics

Irreducible error in other contexts

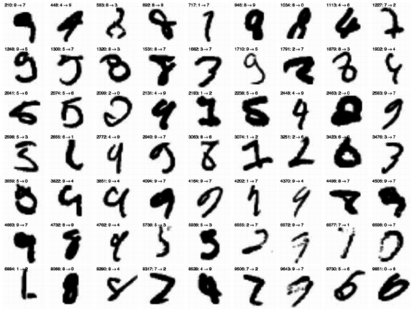
Is a digit a “5” or something else?



How might the concept of irreducible error manifest in this application?

Irreducible error in other contexts

We could know the exact "rules" used to make a "5", but it's possible we encounter examples of "5" that look more like a "6".

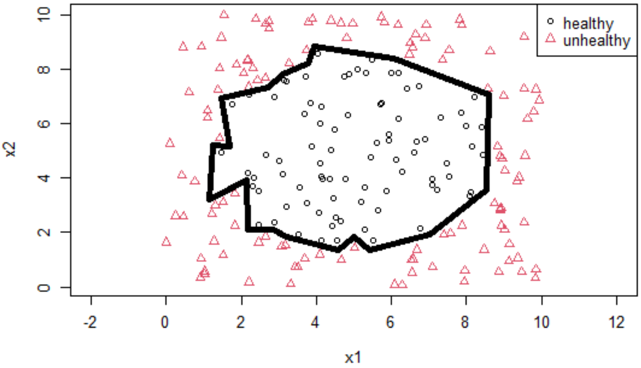


Even state-of-the-art classifiers (which approach the irreducible error rate) incorrectly classify $\sim 0.5\%$ of handwritten digits (source)

Grinnell College
Statistics

Reducible Error

Achieving the best approximation of $f()$ amounts to minimizing *reducible error*. Consider the following classifier:



ıced the error rate to zero?

Training vs. Testing Splits

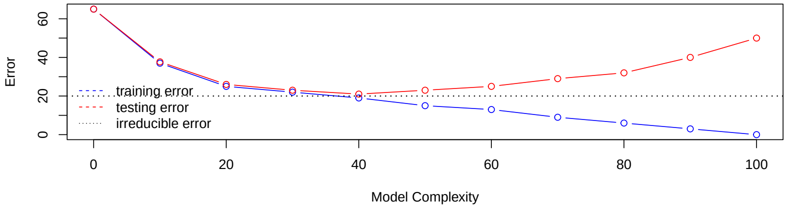
- We aren't really interested in the error rate for *observed examples*
  - Instead, we'd like to minimize reducible error on *new examples* that our model *hasn't seen yet*

Training vs. Testing Splits

- We aren't really interested in the error rate for *observed examples*
  - Instead, we'd like to minimize reducible error on *new examples* that our model *hasn't seen yet*
- Standard protocol is to split the available data into **training** and **testing** sets
  - The training set is used to *learn* a collection of rules
  - The testing set is used to *validate* how well these rules perform on unseen data

Training, Testing, and Error

- Consider a hypothetical example with an irreducible error of “20 units”
  - Training error can be reduced by increasing model complexity (ie: learning more rules)
  - Test error is bounded in probability by the irreducible error



Grinnell College
Statistics

Bias vs. Variance

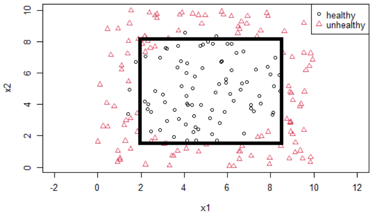
Reducible error can arise in one of two ways: **bias** or **variance**

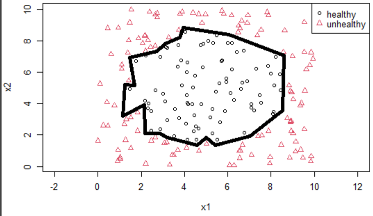
- **Bias** is when a learner lacks the structural flexibility to detect aspects of the true relationship between the predictors and the outcome
- **Variance** is when a learner is overly sensitive to chance artifacts present in the data (ie: the manifestations of irreducible error)

Poor performance due to high bias is called *underfitting*, while poor performance due to high variance is called *overfitting*

Bias vs. Variance

How would you compare the bias and variance of the following models (a rectangle vs. an n-dimensional polygon)?





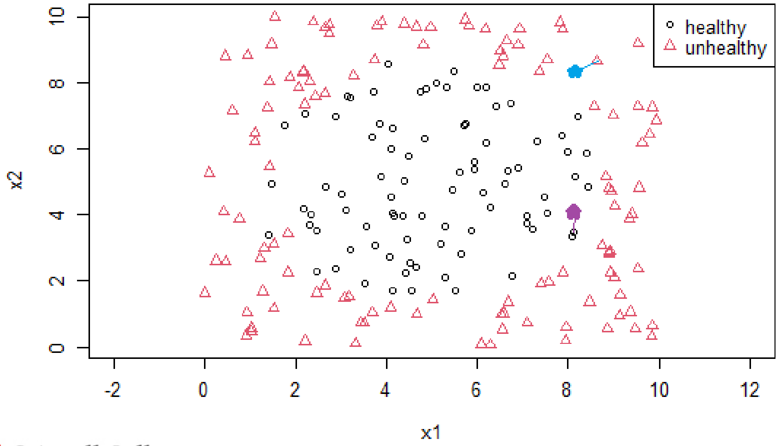
Grinnell College Statistics

Quantifying error

- Our toy example used a binary categorical outcome, a scenario where *classification accuracy* provides a natural way to understand error
  - Later this week we'll consider other ways to measure the error of a classification model
- For a numeric outcome, it's most natural to measure error by summarizing the distances between predicted and observed outcomes:
  - **Root Mean Squared Error**: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
  - **Mean Absolute Error**: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Where $\hat{y}_i$ our model's prediction for the i^{th} data-point

k-nearest neighbors

A simple rule is to classify each new data-point using its *nearest neighbor*, or the observation closest to it's x_2 and x_1 coordinates:



Grinnell College
Statistics

Minkowski distance

To implement this approach, we need to define how to determine the nearest neighbor:

$$d(\mathbf{x}_a, \mathbf{x}_b) = \left(\sum_{j=1}^{p} |x_{a,j} - x_{b,j}|^q \right)^{1/q}$$

- Minkowski distance, $d_{a,b}$, measures the distance between data-points a and b
    - The formula sums *pairwise coordinate differences* across K dimensions
    - The parameter p is chosen by the analyst

Popular distance measures

Two of the most popular choices are $p = 2$ and $p = 1$:

$$d_{\text{euclidean}} = \sqrt{\sum_{j=1}^{p} (x_{a,j} - x_{b,j})^2}$$

$$d_{\text{manhattan}} = \sum_{j=1}^{p} |x_{a,j} - x_{b,j}|$$

When might these measures lead to different neighbors?

Other hyperparameters (weighting)

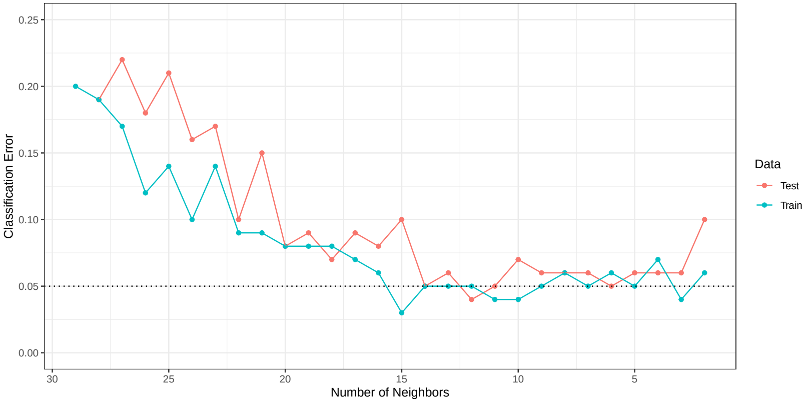
There are two schemes by which neighbors can contribute to a prediction:

1. *Uniform weighting* - each neighbor contributes equally
2. *Distance weighting* - neighbors are weighted by the inverse of their distance to the new data-point, allowing closer neighbors to contribute more strongly

For regression tasks, the prediction is either a simple or weighted average. For classification tasks, the predicted probabilities are given by either simple or weighted voting.

Other hyperparameters (number of neighbors)

Notice the gap in performance on the test vs. training data:



Grinnell College
Statistics

Scaling and standardization

Because KNN relies upon distance calculations, rescaling the predictors is important:

1. **Standardization**:
$$x_i^* = \frac{x_i - \text{mean}(x)}{\text{sd}(x)}$$
2. **Robust scaling**:
$$x_i^* = \frac{x_i - \text{median}(x)}{\text{IQR}(x)}$$
3. **Min-Max scaling**:
$$x_i^* = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$
4. **Max-Absolute scaling**:
$$x_i^* = \frac{x_i}{\max(|x|)}$$

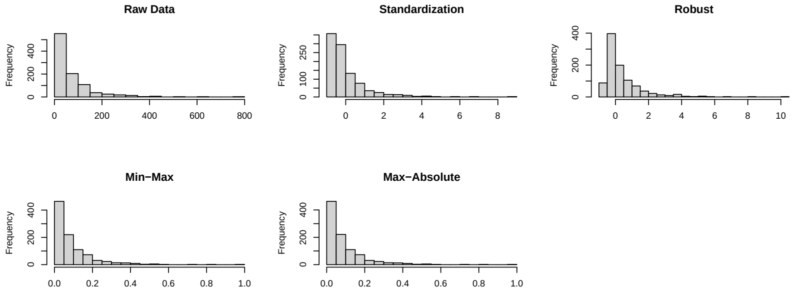
Grinnell College
Statistics

Scaling and standardization

- Standardization forces features to have a *mean of zero* and a *standard deviation of one*
  - Robust scaling forces features to have *a median of zero*, and it can be beneficial for data with large outliers
- Min-Max scaling maps each feature onto a [0,1] interval, which can have computational advantages
  - Max-Absolute scaling is similar to Min-Max scaling, but the output range is [-1,1] and it will *preserve exact zeros* (important for sparse data)

Scaling vs. Normalization

Scaling changes the range of your data, it does not change the distributional shape:



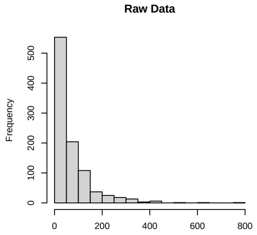
However, the choice of scaler does matter for *k*-nearest neighbors, since dimensions are rescaled individually

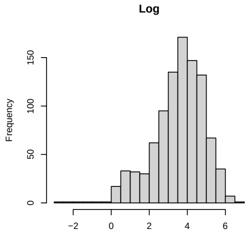
Grinnell College
Statistics

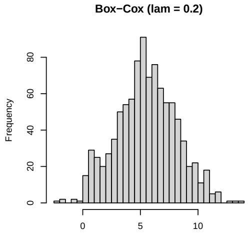
Normalization

If you'd like to change the distributional shape of your data to reduce the effects of skew/outliers, two strategies are:

1. Log-transformation - simply taking the logarithm of each of the variable's values
2. Box-Cox transformation - $x_i^* = \frac{x_i^\lambda - 1}{\lambda}$ for $\lambda \neq 0$ and $X > 0$







Grinnell College
Statistics

Putting it all together

k-nearest neighbors is a simple model that is useful in illustrating the basic workflow of machine learning:

1. Split the data into training and validation sets
2. Set up a data preparation pipeline (ie: rescaling, normalization, dimension reduction, etc.)
3. Determine hyperparamters (ie: weighting, number of neighbors, etc.)
4. Evaluate the final model on the validation set

This week's lab will cover each of these steps in greater detail