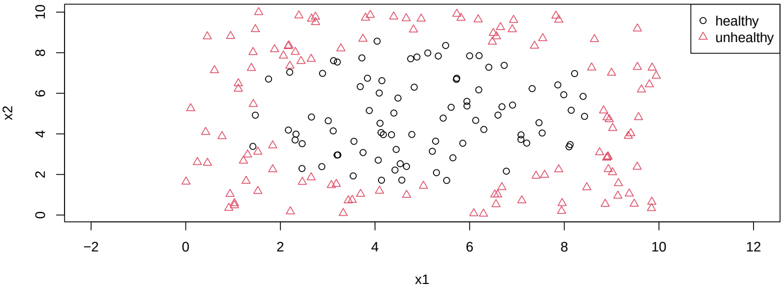
k-Nearest Neighbors

Ryan Miller

Outline

Example Revisited

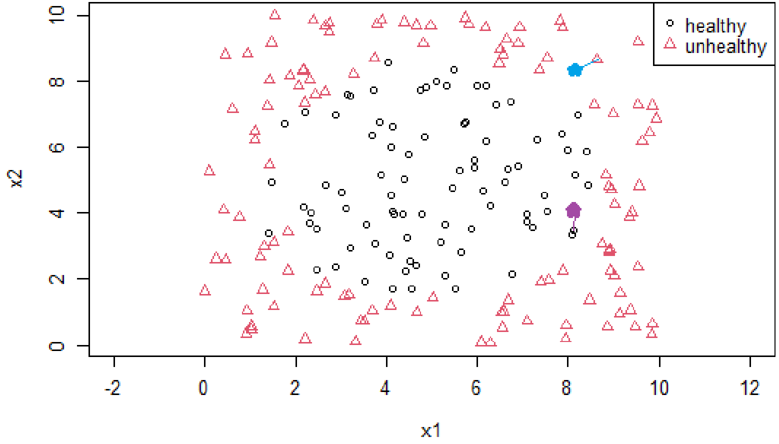
Last week we introduced the following toy example:



Our goal was to learn rules involving x_1 and x_2 that can accurately classify a *new observation* as healthy or unhealthy.

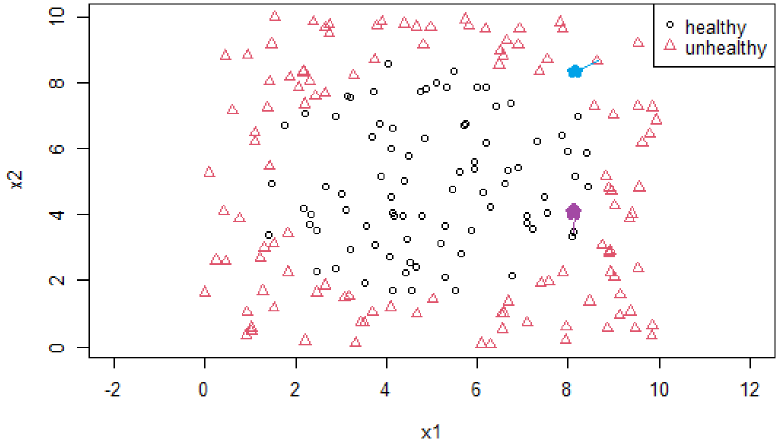
k-nearest Neighbors

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:



k-nearest Neighbors

A new observation at $\{x_1 = 8, x_2 = 4\}$ (purple) is classified as "healthy", while another new observation at $\{x_1 = 8, x_2 = 8\}$ (blue) is classified as "unhealthy":



Minkowski distance

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

$$d_{a,b} = \left(\sum_{j=1}^{K} |x_{a,j} - x_{b,j}|^p \right)^{1/p}$$

- 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 $p = 1$:

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

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

When might these measures lead to different neighbors?

Popular distance measures

- Differences in $x_{a,j} - x_{b,j}$ increase Euclidean distance quadratically, but increase Manhattan distance linearly
  - This means large differences are further magnified by Euclidean distance

Distance and the curse of dimensionality

The *curse of dimensionality* describes a multitude of problems that arise when working with many variables.

> *It's relatively easy to trap a rat inside a pipe (which can move in 1-dimension). It's harder to trap a dog running in a field (which can move in 2-dimensions). It's even harder to trap a bird flying in a field (which moves in 3-dimensions). It's impossible to trap a ghost...*

What problems might arise when using Euclidean distance to find neighbors in a high-dimensional dataset?

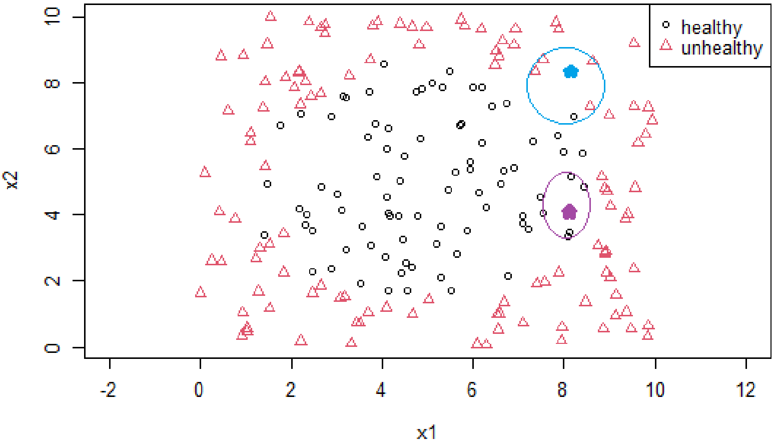
Analogy credit: https://stats.stackexchange.com/questions/169156/explain-curse-of-dimensionality-to-a-child

k-nearest Neighbors

- In our earlier examples, we classified new observations using the *single* nearest neighbor - a *high variance* procedure
  - We can decrease variance (at the expense of introducing additional bias) by using multiple neighbors
  - The *k*-nearest neighbors algorithm aggregates the outcomes of the *k* nearest data-points to generate a prediction for a new observation

k-nearest Neighbors

For a classification task, each of the *k*-nearest data-points contributes to the predicted class:



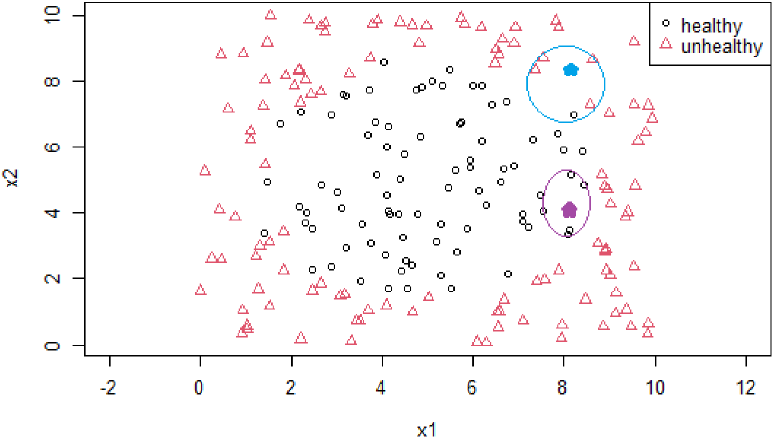
k-nearest Neighbors

There are two schemes by which neighbors can contribute to the predicted class:

1. *Uniform weighting* - the predicted probability of each class equals the proportion of neighbors belonging to that class.
2. *Distance weighting* - neighbors that are weighted by the inverse of their distance, allowing closer data-points to contribute more to the predicted class.

k-nearest Neighbors

Under uniform weighting, we estimate $P(\text{blue} = \text{healthy}) = 4/5$, but under distance weighting we'd estimate $P(\text{blue} = \text{healthy}) > 4/5$ since the unhealthy neighbor is also the furthest



Scaling and standardization

- In our simple example, x_1 and x_2 have similar scales (ie: standard deviations of similar magnitude)
  - In practice, we'll frequently encounter features with different scales
  - Unless rescaled, features on larger scales will have more influence on distance calculations than features on smaller scales

Scaling and standardization

A few popular ways to re-scale data are:

1. **Standardization**:

$$\tilde{x}_i = \frac{x_i - \text{mean}(x)}{\text{sd}(x)}$$

2. **Robust scaling**:

$$\tilde{x}_i = \frac{x_i - \text{median}(x)}{\text{IQR}(x)}$$

3. **Min-Max scaling**:

$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

4. **Max-Absolute scaling**:

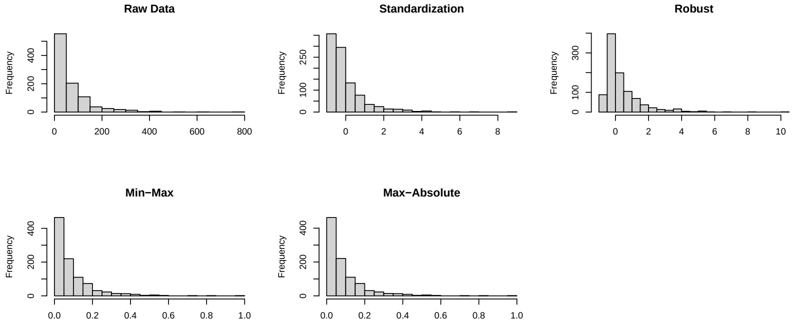
$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$

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:

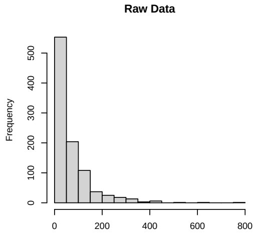


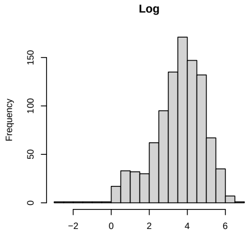
However, the choice of scaler does matter for *k*-nearest neighbors, since dimensions are rescaled individually

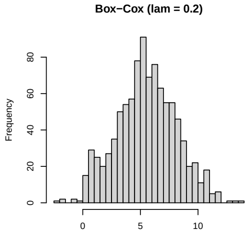
Normalization

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 - $\tilde{X} = \frac{X^\lambda - 1}{\lambda}$ for $\lambda \neq 0$ and $X > 0$







Numeric Outcomes

So far, this presentation has focused exclusively on *classification tasks*, but *k*-nearest neighbors is easily applied to *regression tasks* with the following modifications:

- Outcomes are predicted by the arithmetic average (or distance weighted average) of the numeric outcomes for the identified neighbors
- Model performance is assessed using measures such as *root mean squared error* (RMSE) or *mean absolute error* (MAE)

Putting it all together

To apply the *k*-nearest neighbors machine learning algorithm the following decisions must be made:

1. Whether/how the data should be standardized/scaled and/or normalized
2. How to measure distance
3. The number of neighbors to be used
4. Whether to use uniform or distance weighting

Today's lab will cover Python implementations of each of these; our next lab will cover data-driven methods for making these decisions.