STA 314: Statistical Methods for Machine Learning I

Lecture 9 - Moving beyond linearity, *k*-Nearest Neighbor

Xin Bing

Department of Statistical Sciences
University of Toronto

Moving Beyond Linearity

The linearity assumption in the feature space is almost always an approximation, and sometimes a poor one.

We consider the following extensions to relax the linearity assumption.

- Univariate feature ($p = 1$):
  - Polynomial regression
  - Step functions
  - Regression splines
- Multivariate feature ($p > 1$):
  - Local regression
  - Generalized additive models

Polynomial Regression

- The **polynomial regression**

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \epsilon_i,$$

  where ϵ_i is the error term and $x_i \in \mathbb{R}$.

- Can be fitted by the OLS approach.

- Coefficients themselves are not interpretable; we are more interested in the trend of the fitted function

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \cdots + \hat{\beta}_d x_0^d.$$

- The degree d in practice is typically no greater than 4, and can be chosen via cross-validation.

Polynomial Regression

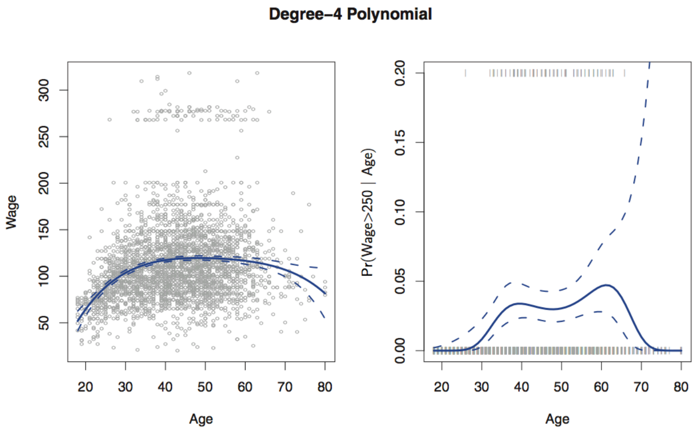
- The polynomial regression can be used for classification as well.
  - For instance, in the logistic regression,

$$\text{logit}\left(\mathbb{P}(Y_i = 1 \mid X_i = x_i)\right) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d.$$

  - Can be fit by maximizing the likelihood.

- However, polynomials have notorious tail behavior – very bad for extrapolation.

The Wage Data



Left: The solid blue curve is a degree-4 polynomial of wage as a function of age, fit by the OLS. The dotted curves are estimated 95 % confidence intervals.
Right: We model the binary event 1{wage > 250} using logistic regression, with a degree-4 polynomial.

Step Functions

- The polynomial regression imposes a global structure on the non-linearity of X.
- The **step function** approach avoids such a global structure by breaking the range of X into bins.
- For pre-specified K cut points $c_1, \dots, c_K$, define

$$
\begin{aligned}
C_0(X) &= 1\{X < c_1\}, \\
C_1(X) &= 1\{c_1 \le X < c_2\}, \\
&\vdots \\
C_K(X) &= 1\{c_K \le X\}.
\end{aligned}
$$

$C_0(X), \dots, C_K(X)$ are in fact $(K+1)$ dummy variables, and they sum up to 1.

Step Functions

- Step function approach assumes

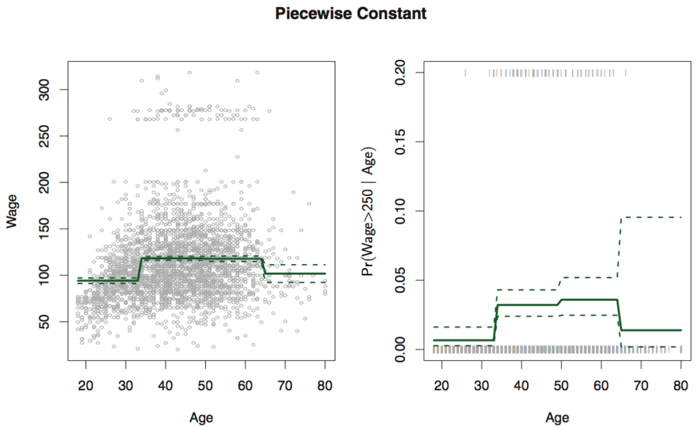
$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \ldots + \beta_K C_K(x_i) + \epsilon_i,$$

  where ϵ_i is the error term. (Note we don't need $C_0(x_i)$ in the model when we also have the intercept term β_0.)

- Can be fitted by the OLS.

- β_j represents the average change in the response Y for $c_j \leq X < c_{j+1}$ relative to $X < c_1$.

The Wage Data



Left: The solid blue curve is a step function of wage as a function of age, fit by least squares. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event wage>250 using logistic regression, with the step function.

Pros and Cons of Step Function

- The step function approach is widely used in biostatistics and epidemiology among other areas, because the model is easy to fit and the regression coefficient has a natural interpretation.
- However, piecewise-constant functions can miss the trend of the true relationship between Y and X. The choice of cut points can be difficult to specify.
- How about combining polynomial and step function?

Piecewise Polynomials

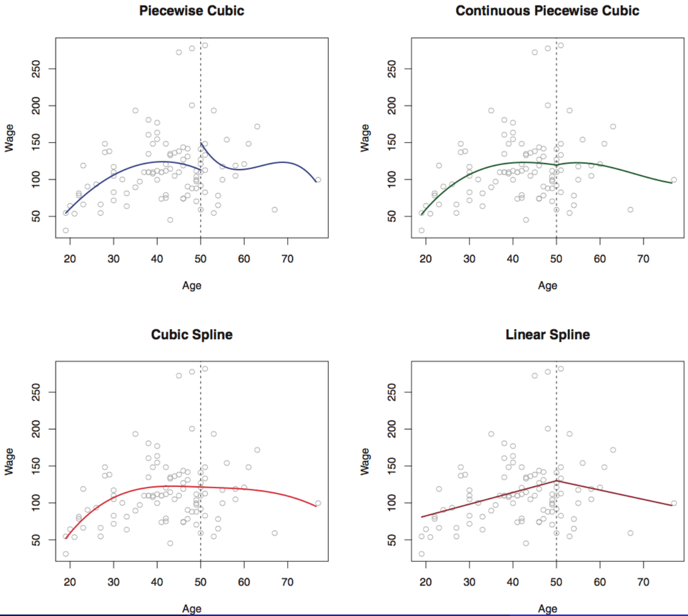
- Instead of a single polynomial in X over its whole domain, we can use different polynomials in different regions:

$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

- The cut point c is called **knot**. Using more knots leads to a more flexible piecewise polynomial.

- In general, if we place K different knots throughout the range of X, then we will end up fitting $(K + 1)$ different cubic polynomials.

The Wage Data



Regression splines

- Better to add constraints to polynomials at the knots for:
  - continuity: equal function values
  - smoothness: equal first and second order derivatives
  - higher order derivatives
- The constrained polynomials are called **splines**. A degree-d spline contains piecewise degree-d polynomials, with continuity in derivatives up to degree $(d-1)$ at each knot.
- How can we construct the degree-d spline?

Linear Splines

- A **linear spline** has piecewise linear functions continuous at each knot. That is, with knots at $\xi_1 < \xi_2 < \cdots < \xi_K$,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 (x_i - \xi_1)_+ \cdots + \beta_{K+1}(x_i - \xi_K)_+ + \epsilon_i,$$

  where

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}.$$

- A basis representation:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+1} b_{K+1}(x_i) + \epsilon_i,$$

  where b_k are basis functions

$$b_1(x_i) = x_i, \quad b_{k+1}(x_i) = (x_i - \xi_k)_+, \quad k = 1, \ldots, K,$$

- Interpretation of β_1: the averaged increase of Y associated with one unit of X for $X < \xi_1$.

Cubic Splines

- A **cubic spline** has piecewise cubic polynomials with continuous derivatives up to order 2 at each knot. That is, with K knots at $\xi_1 < \xi_2 < \cdots < \xi_K$,

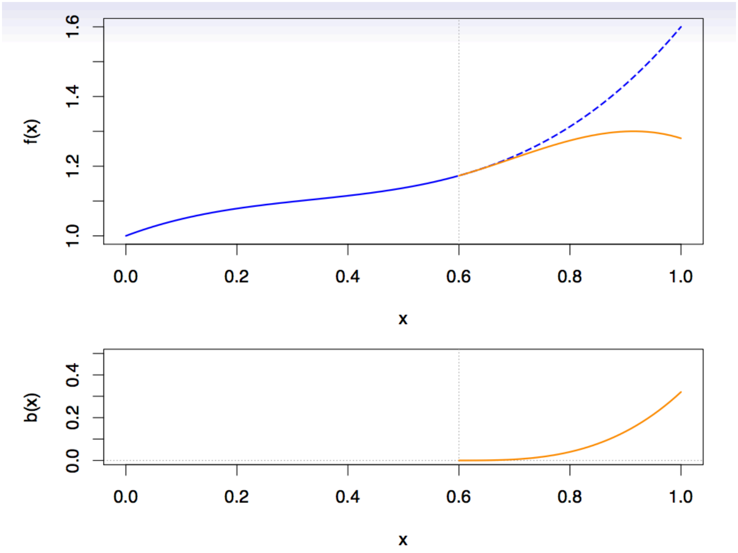
  $$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

  where b_k are basis functions

  $$b_1(x_i) = x_i, \quad b_2(x_i) = x_i^2, \quad b_3(x_i) = x_i^3,$$

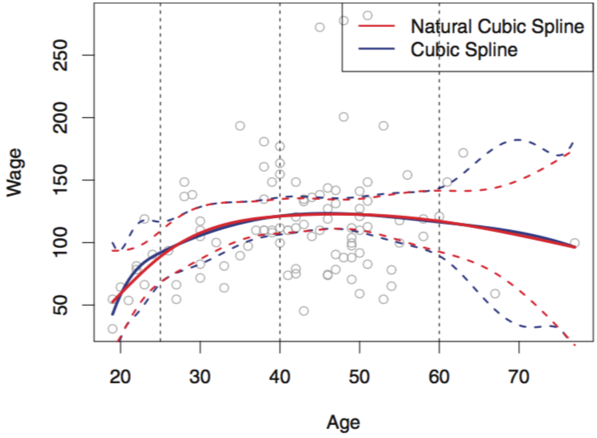
  $$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K.$$

Cubic Splines



Natural Splines

A natural spline is a regression spline with additional boundary constraints: the function is required to be linear at the boundary.

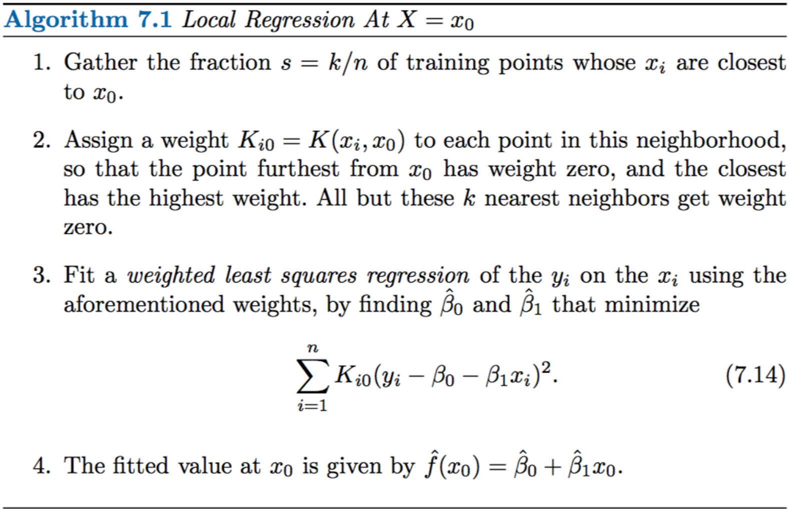


More on splines

- Choosing the number and locations of the knots
  - Typically, we place K knots at certain quantiles of the data or place on the range of X with equal space. Oftentimes, the placement of knots is not very crucial.
  - We use cross-validation to choose K.
- Polynomial regressions and step functions are special cases of splines.
- Another variant: smoothing spline (ISLR 7.5).

Local Regression

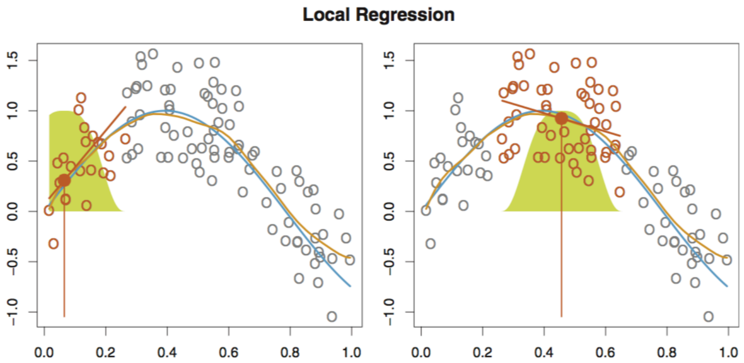
Local regression predicts at a target point x_0 using only the nearby training observations.



Algorithm 7.1 *Local Regression At* $X = x_0$

1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.
2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.
3. Fit a *weighted least squares regression* of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize
$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2. \tag{7.14}$$
4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

Simulated Example



The blue curve is true $f(x)$, and the light orange curve is the local regression $\hat{f}(x)$. The orange points are local to the target point x_0, represented by the orange vertical line. The yellow bell-shape indicates weights assigned to each point. The fit $\hat{f}(x_0)$ at x_0 is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at x_0 (orange solid dot) as the estimate $\hat{f}(x_0)$.

Local Regression

- The size of the neighborhood (fraction s of training data) is a tuning parameter, which can be chosen by cross-validation.
- The weight of each point in the neighborhood needs to be specified.
- When we have two dimensional predictors X_1 and X_2, we can simply use 2-dimensional neighborhoods, and fit bivariate linear regression models using the observations that are near each target point in 2-dimensional space.
- However, local regression can perform poorly if p is much larger than about 3 or 4 (the curse of dimensionality).
- k-Nearest Neighbour is one of the most common local regression approaches. It corresponds to $K(x_i, x_0) = \frac{1}{k}$ and $\beta_1 = 0$.

Generalized Additive Models

- **Generalized additive models** (GAMs) provide a general framework for extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining additivity,

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i.$$

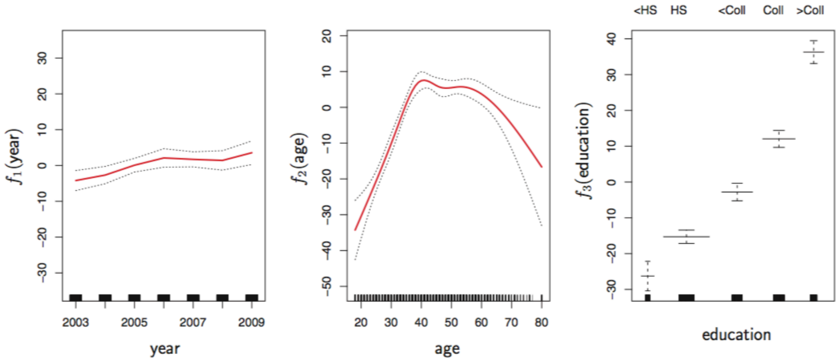
- Each f_k can be linear, polynomials, step function, splines and local regression.
- Can be applied to classification problems.
  - Logistic regression:

$$\text{logit}\left(\mathbb{P}(Y_i = 1 \mid X_i = x_i)\right) = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}).$$

Wage Data

Consider the wage data

$$\text{wage} = \beta_0 + f_1(\text{year}) + f_2(\text{age}) + f_3(\text{education}) + \epsilon.$$



The first two functions are natural splines in year and age. The third function is a step function, fit to the qualitative variable education.

Pros and Cons of GAMs

- GAMs allow us to fit a non-linear f_j to each X_j, so that we can automatically model non-linear relationships that standard linear regression won't be able to capture.
- The non-linear fit can potentially improve prediction accuracy.
- Because the model is additive, we can still examine the effect of each X_j on Y individually while holding all of the other variables fixed.
- It avoids the curse of dimensionality by assuming additivity.
- However, GAMs fail to incorporate the interaction of variables.

A Classical Local Approach: Nearest Neighbors

- Suppose we're given a new feature vector $\mathbf{x} \in \mathbb{R}^p$ we consider classification.
- The idea: find the nearest feature vector to $\mathbf{x}$ in the training set and use its label.
- Can formalize "nearest" in terms of the Euclidean distance

$$\|\mathbf{x}_i - \mathbf{x}_{i'}\|_2 = \sqrt{\sum_{j=1}^{p} (x_{ij} - x_{i'j})^2}$$

Algorithm (1-NN):

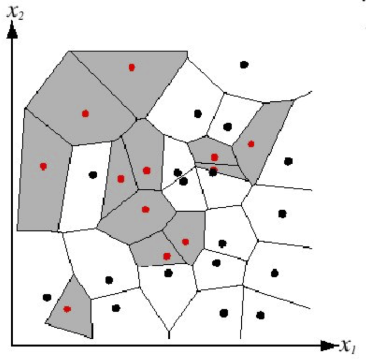
1. Find example $(\mathbf{x}_*, y_*)$ (from the stored training set) closest to $\mathbf{x}$. That is:

$$\mathbf{x}_* = \underset{\mathbf{x}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}}{\operatorname{argmin}} \ \mathrm{distance}(\mathbf{x}_i, \mathbf{x})$$

2. Output y_* as the label

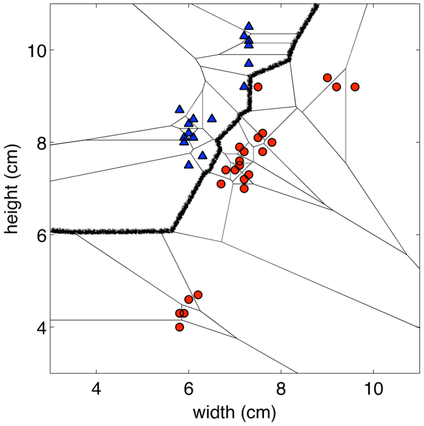
Nearest Neighbors: Decision Boundaries

We can visualize the behavior in the classification setting using a Voronoi diagram.

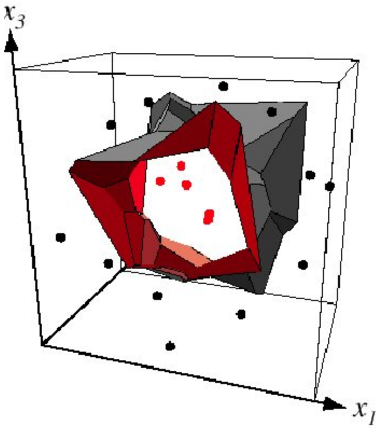


Nearest Neighbors: Decision Boundaries

Decision boundary: the boundary between regions of the feature space assigned to different categories.

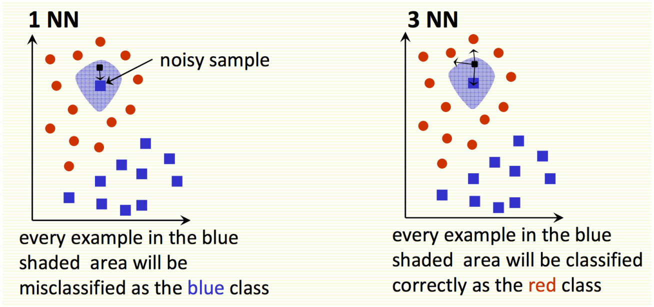


Nearest Neighbors: Decision Boundaries



Example: 2D decision boundary

Nearest Neighbors


1 NN
noisy sample
every example in the blue shaded area will be misclassified as the blue class
3 NN
every example in the blue shaded area will be classified correctly as the red class

- Nearest neighbors sensitive to noise or mis-labeled data ("class noise").
- Solution? Smooth by having k nearest neighbors vote

k-Nearest Neighbors

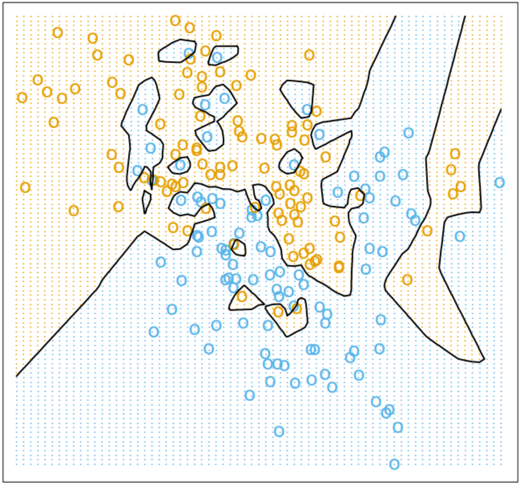
Algorithm (k-NN):

1. Find k data points $(\mathbf{x}_{(1)}, y_{(1)}), \ldots, (\mathbf{x}_{(k)}, y_{(k)})$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$\arg\max_{y \in C} \sum_{i=1}^{k} 1\left\{y = y_{(i)}\right\}.$$

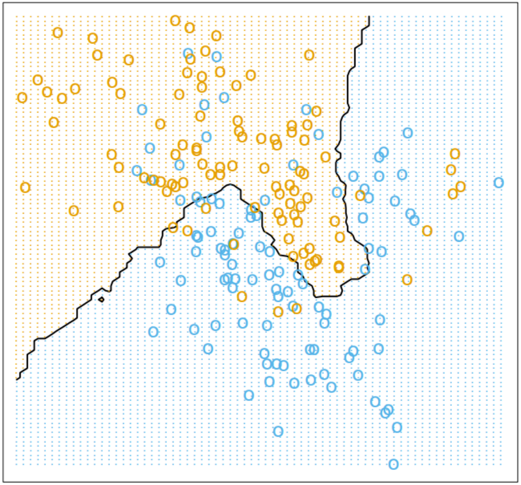
k-NN

k=1



k-NN

k=15



k-NN

Tradeoffs in choosing k

- Small k
  - Good at capturing fine-grained patterns, more flexible decision boundary but high variance
  - Overfitting: may be sensitive to random idiosyncrasies in the training data.
- Large k
  - Less flexible decision boundary and smaller variance
  - Underfitting: may fail to capture important regularities.
- Balancing k
  - Optimal choice of k depends on number of data points n.
  - Nice theoretical properties if

$$k \to \infty, \qquad \text{and} \qquad \frac{k}{n} \to 0 \quad \text{(ESL 2.4)}.$$

  - Rule of thumb: choose k from $[1, \sqrt{n}]$ via cross-validation!

Pitfalls: The Curse of Dimensionality

- *k*-NN suffers the curse of dimensionality!
  - In high dimensions, "most" points are approximately the same distance because they are far away from each other.
- Saving grace: some datasets (e.g. images) may have low intrinsic dimension, i.e. lie on or near a low-dimensional manifold.

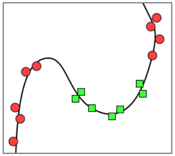
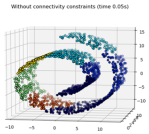
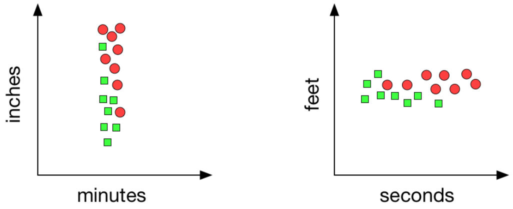



Image credit: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html`

The neighborhood structure depends on the intrinsic dimension.

Pitfalls: Normalization

- Nearest neighbors can be sensitive to the ranges of different features.
- Often, the units are arbitrary:



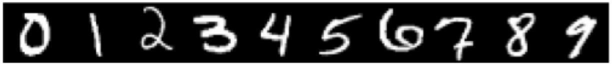
- Simple fix: standardize each dimension to be zero mean and unit variance.
- Caution: depending on the problem, the scale might be important!

Pitfalls: Computational Cost

- Computational cost for training: 0
- Computational cost for classifying test data, per data point (un-modified algorithm)
  - Calculuate p-dimensional Euclidean distances with n data points: $\mathcal{O}(np)$
  - Sort the distances: $\mathcal{O}(n \log n)$
- This must be done for *each* test data point, which is very expensive by the standards of a learning algorithm!
- Need to store the entire dataset in memory!
- Tons of work has gone into algorithms and data structures for efficient nearest neighbors with high dimensions and/or large datasets.

Example: Digit Classification

- Decent performance when lots of data

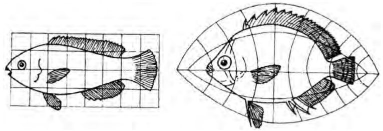


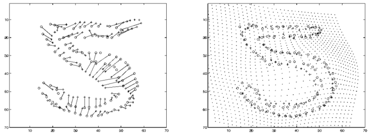
- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

Example: Digit Classification

- Changing the distance measure can really improve k-NN.
- Example: shape contexts for object recognition. In order to achieve invariance to image transformations, they tried to warp one image to match the other image.
  - Distance measure: average distance between corresponding points on *warped* images
- Achieved 0.63% error on MNIST, compared with 3% for Euclidean KNN.
- Competitive with the state of the art at the time, but required careful engineering.





[Belongie, Malik, and Puzicha, 2002. Shape matching and object recognition using shape contexts.]