Lecture 4 - Jan 22

Local Regression and Kernel Smoothing
Radial Basis Function Networks

Recommended Reading

Week 3 Notes on GitHub

Elements of Statistical Learning
2.8.2 Kernel Methods and Local Regression
5.2 Piecewise Polynomials and Splines **(popular LBF expansion)**
6.1-4 Kernel Smoothing Methods **(for Regression)**
6.7 Radial Basis Functions and Kernels

Upcoming Deadlines

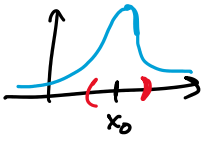
Homework 2 (Jan 27)

Methods that explicitly provide estimates of regression functions or conditional exp. by specifying the nature of the local neighborhood and class of regular functions fitted locally

Local neighborhood is specified by a kernel function $K_\lambda(x_0, x)$ which assigns weights to points x in a region around x_0

Common: Gaussian kernel assigns weights to points that die out exponentially with squared distance

$$K_\lambda(x_0, x) = \frac{1}{\lambda} \exp\left(-\frac{\|x - x_0\|^2}{2\lambda}\right)$$



λ: variance controls width of neighborhood (radius)

We can estimate $f(x_0)$ locally as $f_{\hat{\theta}}(x_0)$ where $\hat{\theta}$ minimizes

$$L(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y - f_\theta(x_i))^2$$

Local Linear Regression

weighted least squares at each input x_0

$$\min_{\alpha(x_0), \beta(x_0)} \sum_{i=1}^{n} \underbrace{K_\lambda(x_0, x_i)}_{\text{not sq.}} \underbrace{(y - \alpha(x_0) - \beta(x_0) x_i)^2}_{\text{sq.}}$$

$\alpha(x_0)$: y-int. $\beta(x_0)$: slope

K_λ: kernel function

weighting matrix

$$\begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad W(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & & \\ & \ddots & \\ & & K_\lambda(x_0, x_n) \end{pmatrix}$$

Define $X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$ $W(x_0) = \begin{pmatrix} K_1(x_0, x_1) & & 0 \\ & K_1(x_0, x_2) & \\ 0 & & \ddots \\ & & & K_1(x_0, x_n) \end{pmatrix}$

$$L(\alpha(x_0), \beta(x_0)) = \left\| \sqrt{W(x_0)} \left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - y \right) \right\|_2^2$$

$$= \left(\sqrt{W(x_0)} \left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - y \right) \right)^T \left(\sqrt{W(x_0)} \left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - y \right) \right)$$

$$= \left(\left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} \right)^T - y^T \right) \left(\sqrt{W(x_0)} \right)^T \sqrt{W(x_0)} \left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - y \right)$$

$$= \left((\alpha(x_0) \ \beta(x_0)) X^T - y^T \right) W(x_0) \left(X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - y \right)$$

$$\vdots$$

$$L(\alpha(x_0), \beta(x_0)) = \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix}^T X^T W(x_0) X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - 2 \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix}^T X^T W(x_0) y + y^T W(x_0) y$$

$$\nabla L(\alpha(x_0), \beta(x_0)) = 2 X^T W(x_0) X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} - 2 X^T W(x_0) y^T = 0$$

$$= X^T W(x_0) X \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} = X^T W(x_0) y^T$$

$$(X^T W(x_0) X)^{-1} (X^T W(x_0) X) \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} = (X^T W(x_0) X)^{-1} X^T W(x_0) y^T$$

Fit different simple models at each query point x_0 using only nearby points such that the estimated fune $\hat{f}(X)$ is smooth in $\mathbb{R}^d$ using a weighting function (kernel) assigning a weight to x_i based on distance to x_0

Parameter λ frequently represents width of the neighborhood and must be learned

6.1 1D Kernel Smoothers

Epanechnikov quadratic kernel $K_\lambda(x_0,x) = D\left(\frac{|x-x_0|}{\lambda}\right)$

with $D(t) = \begin{cases} \frac{3}{4}(1-t^2), & \text{if } |t|\le 1 \\ 0, & \text{else} \end{cases}$ tri-cube: $D(t) = \begin{cases} (1-|t|^3)^3, & \text{if } |t|\le 1 \\ 0, & \text{else} \end{cases}$

More generally, $K_\lambda(x_0,x) = D\left(\frac{|x-x_0|}{h_\lambda(x_0)}\right)$

Details to determine

(1) λ - smoothing parameter
Large → lower variance (average over more observations)
but more bias (true fune constant in windows)

(4) Weights

(5) Boundary issues

(6) Ep. kernel has compact support, Gaussian not

6.1.1 Local Linear Regression

Weighted least squares at each target point x_0

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0,x_i)\underbrace{\left(y_i-\alpha(x_0)-\beta(x_0)x_i\right)^2}_{L(\alpha(x_0),\beta(x_0))}$$

where $\hat{f}(x_0)=\hat{\alpha}(x_0)+\hat{\beta}(x_0)x_0$

$$L(\alpha(x_0),\beta(x_0)) = \left\|\sqrt{W(x_0)}\left(B\begin{pmatrix}\alpha(x_0)\\ \beta(x_0)\end{pmatrix}-y\right)\right\|_2^2 = \left(\sqrt{W(x_0)}\left(B\begin{pmatrix}\alpha(x_0)\\ \beta(x_0)\end{pmatrix}-y\right)\right)^T\left(\sqrt{W(x_0)}\left(B\begin{pmatrix}\alpha(x_0)\\ \beta(x_0)\end{pmatrix}-y\right)\right)$$

Define $B = \begin{pmatrix}1 & x_1\\ 1 & x_2\\ \vdots & \vdots\\ 1 & x_n\end{pmatrix} \leftarrow b(x_1)^T \leftarrow b(x_2)^T \leftarrow b(x_n)^T$ regression matrix

$$W(x_0) = \begin{pmatrix}K_\lambda(x_0,x_1) & & \\ & K_\lambda(x_0,x_2) & \\ & & \ddots \\ & & & K_\lambda(x_0,x_n)\end{pmatrix}$$ kernel weights

$$L(\alpha(x_0),\beta(x_0)) = \left(\left(B\Theta - y\right)^T\sqrt{W(x_0)}^T\right)\left(\sqrt{W(x_0)}B\Theta - \sqrt{W(x_0)}y\right)$$

$$= \left(\Theta^T B^T\sqrt{W(x_0)}^T - y^T\sqrt{W(x_0)}^T\right)\left(\sqrt{W(x_0)}B\Theta - \sqrt{W(x_0)}y\right)$$

$$= \Theta^T B^T\sqrt{W(x_0)}^T\sqrt{W(x_0)}B\Theta$$

$$- \underbrace{\Theta^T B^T\sqrt{W(x_0)}^T\sqrt{W(x_0)}y}_{1\times 2\ \ 2\times N\ \ N\times N\ \ N\times N\ \ N\times 1} - \underbrace{y^T\sqrt{W(x_0)}^T\sqrt{W(x_0)}B\Theta}_{1\times N\ \ N\times N\ \ N\times N\ \ N\times 2\ \ 2\times 1}$$ Same

$$+ y^T\sqrt{W(x_0)}^T\sqrt{W(x_0)}y$$

$$= \Theta^T B^T W(x_0)B\Theta - 2\Theta^T B^T W(x_0)y + y^T W(x_0)y$$

$$L(\theta) = \theta^T B^T W(x_0) B\theta - 2\theta^T B^T W(x_0) y + y^T W(x_0) y$$

$$\nabla L(\theta) = 2 B^T W(x_0) B\theta - 2 B^T W(x_0) y = 0$$

$$(B^T W(x_0) B)^{-1} B^T W(x_0) B\theta = (B^T W(x_0) B)^{-1} B^T W(x_0) y$$

$$\theta = (B^T W(x_0) B)^{-1} B^T W(x_0) y$$

$$\Rightarrow \hat{f}(x_0) = (1 \; x_0)\theta = (1 \; x_0)(B^T W(x_0) B)^{-1} B^T W(x_0) y$$

$$= \sum_{i=1}^{N} l_i(x_0) y_i \quad \leftarrow \text{linear in } y_i\text{'s}$$

$l_i(x_0)$: equivalent kernel

Selecting Kernel Width

Monday, January 25, 2021 11:54 AM

6.2 Selecting the Width of the Kernel

In each of the kernels K_λ, λ is a parameter that controls its width:

- For the Epanechnikov or tri-cube kernel with metric width, λ is the radius of the support region.
- For the Gaussian kernel, λ is the standard deviation.

There is a natural bias–variance tradeoff as we change the width of the averaging window, which is most explicit for local averages:

- If the window is narrow, $\hat{f}(x_0)$ is an average of a small number of y_i close to x_0, and its variance will be relatively large—close to that of an individual y_i. The bias will tend to be small, again because each of the $E(y_i) = f(x_i)$ should be close to $f(x_0)$.
- If the window is wide, the variance of $\hat{f}(x_0)$ will be small relative to the variance of any y_i, because of the effects of averaging. The bias will be higher, because we are now using observations x_i further from x_0, and there is no guarantee that $f(x_i)$ will be close to $f(x_0)$.

Similar arguments apply to local regression estimates, say local linear: as the width goes to zero, the estimates approach a piecewise-linear function that interpolates the training data[1]; as the width gets infinitely large, the fit approaches the global linear least-squares fit to the data.

Generalization to $d > 1$ is natural

- Local linear regression fits hyperplanes locally via weighted least squares (preferred due to performance on boundaries)

Let $b(X)$ be a vector of polynomial terms in X of max degree P.

if $d=2, p=1 \rightarrow b(X) = (1, X_1, X_2)$

$d=2, p=2 \rightarrow b(X) = (1, X_1, X_2, X_1^2, X_2^2, X_1X_2)$

For $x_0 \in \mathbb{R}^d$, solve $\min_{\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - b(x_i)^T \beta(x_0))^2$

$\hookrightarrow \hat{f}(x_0) = b(x_0)^T \hat{\beta}(x_0)$

Typically, K is a radial function like Ep./tri-cube

$$K_\lambda(x_0, x) = D\left(\frac{\|x - x_0\|_2}{\lambda}\right)$$

It is common to standardize predictors, e.g. to $\sigma = 1$, before smoothing

Higher dims have more problems on boundaries as freq of pts close

Higher dims have more problems on boundaries as proximity to boundary grows

↳ local polynomial regression performs boundary correction to the desired order in any dimension

↳ Local reg. less useful for $d > 3$

Review of Kernel Regression

Lecture 6 (Mon, Jan 25) Kernel Regression

Please see the notes in Canvas for details, but we derived the parameters of a local linear regression model in the neighborhood of the point x_0 for $d = 1$ by minimizing the weighted squared error loss function,

$$L(\alpha(x_0), \beta(x_0)) = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - \alpha(x_0) - \beta(x_0)x_i)^2$$

The function K_λ is called a kernel, or weighting, function. Intuitively, it measures how similar two points are. If x_i and x_0 are close, the kernel function will be large, so x_i will have a larger impact on the local regression model. A common kernel function is the Gaussian kernel,

$$K_\lambda(x_0, x) = \frac{1}{\lambda}\exp\left(-\frac{\|x - x_0\|_2^2}{2\lambda}\right)$$

We found the parameters of the unknown function f in the vicinity of x_0 to be

$$\begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} = (X^T W(x_0) X)^{-1} X^T W(x_0) y,$$

where

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad W(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & & & \\ & K_\lambda(x_0, x_2) & & \\ & & \ddots & \\ & & & K_\lambda(x_0, x_n) \end{pmatrix}$$

where $W(x_0)$ is a diagonal matrix, meaning all entries except the diagonal entries are 0.

And, so, the local linear approximation of the unknown function f at x_0 is

$$\hat{f}(x_0) = \begin{pmatrix} 1 & x_0 \end{pmatrix} (X^T W(x_0) X)^{-1} X^T W(x_0) y$$

Implementing Kernel Regression

Let's write a class that will carry out kernel regression to construct local linear regression models. The class will take the $\boldsymbol{X}$ data, $\boldsymbol{y}$ data, and kernel function as inputs. But, first, let's import some basic things like `NumPy`, `pandas`, `matplotlib`, and some quality metrics for regression models from `scikit-learn`.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

Now, the class.

```
def GaussianKernel(x0, x, lam):
    return (1/lam) * np.exp(-np.linalg.norm(x0 - x) / (2 * lam))

class KernelRegression:
    def __init__(self, kernel_function, lam, fit_intercept = True):
        self.kernel_function = kernel_function
        self.fit_intercept = fit_intercept
        self.lam = lam

    def predict(self, x0, X, y):
        # find the number of X points
        n = X.shape[0]

        # add a column of ones if needed
        if self.fit_intercept:
            X = np.hstack((np.ones([n,1]), X))

        # construct the kernel matrix
        kernel = np.zeros([n, n])

        # populate the kernel matrix
        for i in range(n):
            kernel[i][i] = self.kernel_function(x0, X[i,:], self.lam)

        return np.array([1, np.float(x0)]) @ np.linalg.inv((X.T @ kernel @ X)) @ X.T @ kernel @ y
```

Let's try it with the shampoo dataset from last week. First, read in the data

```
# read the shampoo sales dataset
data = pd.read_csv('data/shampoo.csv')

# save the targets
y = data['Sales'].to_numpy()

# make a column vector of 0s with n elements
X = np.zeros([y.shape[0], 1])

# convert the vector to (0, 1, 2, ..., n)
X[:,0] = [i for i in range(y.shape[0])]

# split the data into train and test sets
(trainX, testX, trainY, testY) = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

Next, let's run the kernel regression model at each x value in a reasonable range using the Gaussian kernel with $\lambda \in \{0.1n : n = 1, \ldots, 100\}$ and plot each.

```
lamValues = np.linspace(0.025, 10, 100)

M = lamValues.shape[0]

# allow multiple plots
fig, axes = plt.subplots(nrows = M + 1, figsize = (10, 4*M))

trainMAE = []
testMAE = []

for (j, lam) in enumerate(lamValues):
    model = KernelRegression(GaussianKernel, lam)

    # compute the model coordinates
    xModel = [i for i in range(int(np.round(1.1 * np.max(X))))]
    yModel = [model.predict(i, trainX, trainY) for i in xModel]

    # plot the model
    label = 'Kernel Smoothing for lambda = ' + str(np.round(lam, 2))
    axes[j].scatter(X, y)
    axes[j].plot(xModel, yModel, 'r', label = label)
    axes[j].legend()

    # apply the basis functions to the test data and predict with the model
    trainPredictions = [model.predict(i, trainX, trainY) for i in trainX]
    testPredictions = [model.predict(i, trainX, trainY) for i in testX]

    # compute the training and test mean absolute error
    trainError = mean_absolute_error(trainY, trainPredictions)
    testError = mean_absolute_error(testY, testPredictions)

    # save the training and test mean absolute error
    trainMAE.append(trainError)
    testMAE.append(testError)

    # return quality metrics
    print('lambda:', np.round(lam, 3), '\t\tr^2:', np.round(r2_score(trainY, trainPredictions), 3),
          '\t\ttrain MAE:', np.round(trainError, 3), '\t\ttest MAE:', np.round(testError, 3))

# plot the errors
axes[M].plot(range(M), trainMAE, label = 'Training Mean Absolute Error')
axes[M].plot(range(M), testMAE, label = 'Testing Mean Absolute Error')
axes[M].legend()
```

As you can see, with a smaller λ value, the fitted curve has some very sharp curves as the nearby points exert a pulling force on the locally linear curve. But, as we increase λ, the more smoothing we see in the predicted curve--as points further away from the input x_0 have more impact. Increasing λ results in smoother and smoother curves (hence, the name kernel *smoothing*), which become nearer to a linear fit.

Beyond that, a larger λ continually increases the training error and reduces the r^2, but the testing error, the more important measure, is reduced to at $\lambda = 2.554$, but then starts to increase.

Radial Basis Functions and Kernels

In Chapter 5, functions are represented as expansions in basis functions: $f(x) = \sum_{j=1}^{M} \beta_j h_j(x)$. The art of flexible modeling using basis expansions consists of picking an appropriate family of basis functions, and then controlling the complexity of the representation by selection, regularization, or both. Some of the families of basis functions have elements that are defined

Kernel methods achieve flexibility by fitting simple models in a region local to the target point x_0. Localization is achieved via a weighting kernel K_λ, and individual observations receive weights $K_\lambda(x_0, x_i)$.

Radial basis functions combine these ideas, by treating the kernel functions $K_\lambda(\xi, x)$ as basis functions. This leads to the model

Ch 5: $f(x) = \sum_{j=1}^{M} \beta_j h_j(x)$ expansion in basis functions

Ch 6: Fitting simple models locally by weighting kernels

Radial basis functions combine these ideas by treating kernel functions $K_\lambda(\xi, x)$ as basis functions

$$f(x) = \sum_{j=1}^{M} K_{\lambda_j}(\xi_j, x)\beta_j = \sum_{j=1}^{M} D\left(\frac{\|x-\xi_j\|}{\lambda_j}\right)\beta_j$$

scale parameter — prototype parameter — Gaussian popular

Several methods to learn $\lambda_j, \xi_j, \beta_j$ $(j=1,\dots,M)$. We focus on least squares + Gaussian kernel

→ ... SSE w.r.t. all parameters

① Optimize SSE w.r.t. all parameters

$$\min_{\{\lambda_j, \xi_j, \beta_j\}_{j=1}^{m}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{m} \beta_j \exp\left\{ - \frac{(x_i - \xi_j)^T (x_i - \xi_j)}{\lambda_j^2} \right\} \right)^2$$

↑

RBF network, $\xi_j + \lambda_j$ weights

Loss not convex, multiple minima exist → algorithm

② Estimate $\{\lambda_j, \xi_j\}$ separately from β_j. Given λ_j, ξ_j, finding β is OLS.

Could choose λ_j, ξ_j unsupervised using X distribution

↳ e.g. fit Gaussian mixture model to x_i → ξ_j (center) + λ_j (scale)

↳ e.g. clustering to locate prototypes ξ_j + treat $\lambda_j = \lambda$ as a hyper-parameter

$$\hat{f}(x_i) = \theta_0 + \sum_{j=1}^{m} \theta_j K_{\lambda_j}(\xi_j, x_i)$$ — RBF Network

Can we write it in matrix form?

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$X_K = \begin{pmatrix} 1 & K_{\lambda_1}(\xi_1, x_1) & \cdots & K_{\lambda_m}(\xi_m, x_1) \\ 1 & K_{\lambda_1}(\xi_1, x_2) & \cdots & K_{\lambda_m}(\xi_m, x_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & K_{\lambda_1}(\xi_1, x_n) & \cdots & K_{\lambda_m}(\xi_m, x_n) \end{pmatrix}$$

$$\hat{y} = X_K \theta \qquad \text{(predicted } y\text{)}$$

A radial function of x_i and ξ_j depends on $\|x_i - \xi_j\|$, but not x_i or ξ_j individually

The most popular is the Gaussian kernel

The most popular is the [illegible]

$$K_{\lambda_j}(\xi_j, x_i) = \frac{1}{\lambda_j} \exp\left(-\frac{\|x_i - \xi_j\|^2}{2\lambda_j^2}\right)$$

note each ξ_j and x_i is in $\mathbb{R}^d$

Sum of squares loss function

$$L(\theta, \lambda, \xi) = \|X_K \theta - y\|^2$$

unfortunately, X_K depends on $\lambda = (\lambda_1, \ldots, \lambda_m)$

$$\xi = \begin{pmatrix} \xi_1^T \\ \xi_2^T \\ \vdots \\ \xi_m^T \end{pmatrix}$$

so, minimizing L with respect to θ and λ and ξ is not nearly as easy as with LBF expansions

Further, L is not convex, so it may have multiple critical values, so, even if we could solve $\nabla L(\theta, \lambda, \xi) = 0$, it would not be assured to be the global minimum.

it would not be assured