Locally Weighted Regression

J.R. Powers-Luhn

***Abstract*—A nonlinear regression technique called locally weighted regression is examined. This method fits linear-in-parameters curves to small regions around a prediction point. The importance of the training points to the model is scaled by a kernel function that operates on the distance between each training point and the point being predicted. Models are examined for a range of kernel width and polynomial order with the best model being selected by cross validation. A weighted regression model was generated that predicted body fat percentage to a root mean squared error of $7.78\,\%$ body fat.**

I. Introduction

Regression techniques examined to this point have sought to produce a single model that describes a phenomenon across the training domain. This assumption is attractive in that the models produced may reflect physical relationships present in the system being modeled, but linear models that make this assumption may be insufficiently complex to predict accurately across the training domain. Locally weighted regression (LWR) instead approximates the training data as a series of connected linear relationships–that there exists some interval $\mathrm{d}\vec{x}$ small enough that in that region a linear model is a close enough approximation. This trades physical meaning of the model for accuracy of prediction. It also introduces additional free parameters that must be selected via cross validation.

Linear (and linear-in-parameters) regression models examine an entire training set and attempt to fit a single model to it. A single computationally intense calculation (inverting $\mathbf{X}^T\mathbf{X}$, with the possible addition of some regularization term) is performed to determine regression coefficients, which can then be applied to future predictors. Linear models are limited to linear functions, however. No linear regression can fit non-linear (in parameters) functions, which is unfortunate as a number of these functions (e.g. $\sin$, $\exp$, and $\log$) appear in natural processes.

These functions can be approximated as a series of locally linear functions, assuming that the underlying processes are continuous. By adding an importance or "weight" term to represent how representative a training point is of the area around the region being predicted, a locally representative model is generated. It is intuitively reasonable to assume that distant points (determined by the difference in the values of their associated predictor variables) should be less represented in the model and that the importance should go to zero as the distance measure approaches infinity. This allows for broad latitude in the selection of the weight function and the distance function. Any function that follows the following criteria [1] is suitable:

- $w(x) > 0$ for $|x| < 1$,
- $w(x) = w(-x)$, and
- $w(x)$ is non-increasing for $x \geq 0$.

Cleveland identified a fourth criterion (that $w(x) = 0$ for $|x| \geq 1$) which does not apply to Gaussian weighting function. This method was not practical until the advent of modern computers. Since a (possibly nonlinear) weight calculation must be performed for each point to be predicted and training point, it was not feasible to perform this calculation by hand for real data sets. Now these computations can be performed relatively quickly, making the practical use of this method possible.

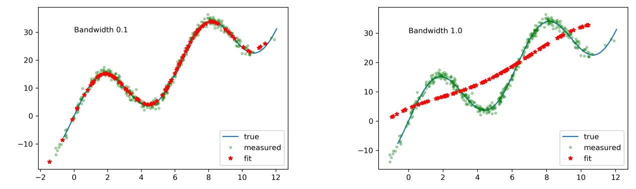
II. Methodology

LWR also changes the computational cost of the modeling process. In previous linear regression techniques the training data is used to generate regression coefficients and can then be discarded. The cost of making new predictions is low, as it simply involves multiplying the inputs by the coefficients. In LWR, the coefficients of the regression model depend on the inputs. The weights must be generated as a function of the distance between the point to be predicted and the training data. A kernel function is then applied to this distance, generating the weights (as in equation 1). This means that the model takes up more computer storage (since all of the training data must be stored) and the computational cost of making predictions is $\mathcal{O}(n_t n_p)$–it scales as the product of the number of training points and the number of prediction points. This shows another trade off in LWR: storing more training data may increase the variance of the model (by representing more regions of the training space), but it also increases the cost of making new predictions.

$$\vec{w} = K(d(\mathbf{X}_t, \vec{x}), b) \tag{1}$$

To implement locally weighted regression it is necessary to choose three parameters: the distance measurement to use, the convolution kernel, and the bandwidth of that kernel. For the purposes of this paper only one distance measure (the ℓ_2 norm, or euclidean distance) and convolution kernel (the Gaussian distribution, equation 2) were examined. Smaller bandwidths effectively shrink the size of the training set. If the bandwidth is too small, local noise could overwhelm the underlying physical process. Conversely, if the bandwidth is too large, the model becomes biased and under fits the data. An infinite bandwidth would weight all training points equally, producing the same result as a linear regression.

For models with input dimensions greater than one, the scale of each input is likely to vary (for example, one input might be in Pa while another might be in $\mathrm{\mu m}$). The varying scales would skew the distance calculation and underweight inputs with large scales. In order to counteract this effect, the input and prediction data were scaled to unit variance before performing any regression calculation.



(a) Bandwidth of 0.1 (b) Bandwidth of 1.0

Fig. 1: Comparison of the impact of bandwidth on locally weighted regression. These are first-order LWR fits with bandwidth of 0.1 (left) and 1.0 (right). While the figure on the right shows some response to the sine curve, it primarily fits the underlying linear trend. The figure on the right fits the model more precisely. Smaller bandwidths (not shown) would begin to over fit the noise present in the data.

$$w_i = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{||r||^2}{2\sigma^2}\right\} \tag{2}$$

As shown in figure 1, the selection of bandwidth has a direct effect on the model's bias vs. variance trade off. In order to select the best bandwidth, a cross validation strategy is employed. The training data is used to generate models with varying parameters and the error from the test set is compared.

A. Kernel regression

The simplest form of LWR is a 0-th order regression or weighted moving average. The value of the dependent variables in the training set are multiplied by the weights (normalized to sum to one) and the sum of these values is the prediction ($\sum_i w_i x_i^t$).

B. Locally weighted regression

A more complex method for fitting the data is to fit higher order polynomial (order one or greater) to the weighted data. The linear regression equation is modified by inserting

$$\vec{b} = (\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}^T\mathbf{W}\vec{y} \tag{3}$$

The $\mathbf{W}$ matrix in equation 3 is a diagonal matrix with the diagonal elements having the values $\sqrt{w_i}$ that has the effect of increasing the conditional number of the term to be inverted. This makes the equation more sensitive to small perturbations. This is counteracted by the fact that the data being inverted is more representative of the shape of the equation in the area around the training points. This trade off is balanced using cross validation to determine the generality of the models for different bandwidth parameters. As such, models with several different bandwidths (0.15, 0.25, 0.50, 0.75, 1.00, 1.50 and 2.00) were trained on the training set and evaluated using the testing set.

C. Body Composition

A data set of body measurements (body fat percentage, age, weight, height, adiposity index, and ten circumference measurements) was obtained [2]. Locally Weighted Regression was used to generate models to predict body fat percentage from the other predictor variables.

III. Results

The RMSE of the models examined ranged from 7.48 to 18.73% (full results in table I). As in pure regression, adding higher order polynomial terms increases the ability of the model to over fit the data. This is reflected in the worse performance of the order two models. This is corrected by increasing the bandwidth, but this could come at a cost of a loss of local predictive power.

TABLE I: Test set RMSE for models with varying order and bandwidth. Models with order zero represent kernel regression, or a weighted moving average.

Order	Bandwidth	RMSE (% bodyfat)
0	0.15	11.13
	0.25	11.07
	0.50	8.97
	0.75	7.99
	1.00	7.64
	1.50	7.50
	2.00	7.48
1	0.15	15.64
	0.25	7.53
	0.50	7.48
	0.75	7.95
	1.00	8.07
	1.50	8.08
	2.00	8.14
2	0.15	9.05
	0.25	18.73
	0.50	13.89
	0.75	12.68
	1.00	13.11
	1.50	8.93
	2.00	7.65

A. Model Selection

The model selected as best was a first order weighted linear regression with a bandwidth of 0.50. This number is dimensionless since it represents the standard deviation of the distance between two scaled (and therefore unitless) vectors. It performed as well as a kernel regression with a bandwidth of two but was selected over that model due to it being slightly more representative of physical trends in the area around the predicted point.

The selected model was evaluated on the validation set data, producing an RMSE of 7.78% body fat.

IV. Conclusions

A model was generated to predict body fat percentage from body composition predictors using locally weighted regression. The error of this model was 7.78% body fat. This compared unfavorably to previously generated models on these data. The error was greater than that produced by a partial least squares model (4.35%), principal component regression (4.72%), a best-guess linear regression (4.79%), and a ridge regression model (4.34%).

The strength and weakness of this modeling technique is that it requires no physical understanding of the underlying data. In the presence of enough training data (enough such that the mean of the noise is zero in each local region) it is possible to select a small enough bandwidth to perfectly predict the data. One purpose of generating models, however, is to understand the underlying processes that produce the measured results. This technique does not provide anything in the way of physical understanding. While that does not detract from the utility of this technique, it should be approached with this limitation in mind.

References

[1] William S Cleveland. “Robust locally weighted regression and smoothing scatterplots”. In: *Journal of the American statistical association* 74.368 (1979), pp. 829–836.

[2] K W Penrose, A G Nelson, and A G Fisher. “Generalized Body Composition Prediction Equation For Men Using Simple Measurement Techniques”. In: *Medicine & Science in Sports & Exercise* 17.2 (1985). ISSN: 0195-9131.

V. Appendix

Python code used to perform calculations and generate graphics.

```
import numpy as np
from scipy.stats import norm
from scipy.spatial.distance import cdist
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from utilities import train_test_val_split, load_matlab_data, rmse
from itertools import product
import pandas as pd


x, y = load_matlab_data("data/hwkdataNEW.mat")
xtr, ytr, xts, yts, xv, yv = train_test_val_split(x, y, seed=3)

class LocallyWeightedRegression(BaseEstimator, RegressorMixin):
    def __init__(self, bandwidth=0.25, order=1):
        self.bandwidth = bandwidth
        self.order = order

        # Save the scalers
        self.xscaler_ = None
        self.yscaler_ = None

        # Save training data
        self.xs_ = None
        self.ys_ = None

    def fit(self, x, y):
        self.xscaler_ = StandardScaler().fit(x)
        self.yscaler_ = StandardScaler().fit(y)

        self.xs_ = self.xscaler_.transform(x)
        self.ys_ = self.yscaler_.transform(y)

        return self

    def predict(self, x):
        xs = self.xscaler_.transform(x)
        results = []

        for x in xs:
            # Calculate the distances
            ds = cdist(self.xs_, np.atleast_2d(x))

            # Calculate the weights
            ws = norm.pdf(ds, scale=self.bandwidth)

            # Normalize the weights
            ws /= sum(ws)

            # Convert to 1d array
            ws = ws[:,0]

            # Add in the polynomial terms
            p = PolynomialFeatures(self.order)
            xt = p.fit_transform(self.xs_)

            # Perform the regression to get the coefficients
            l = LinearRegression()
            l.fit(xt, self.ys_, sample_weight=ws)

            # Calculated the predicted value
            res = l.predict(p.transform(x.reshape((1, -1))))

            # Add that to the results array
            results.append(res)

        res_np = np.array(res).reshape(-1, 1)

        return self.yscaler_.inverse_transform(res_np)

    def score(self, x, y):
        yp = self.predict(x)
        return -rmse(y, yp)

r_dict = {
    'Bandwidth': [],
    'Order': [],
    'RMSE (% bodyfat)': []
}
bandwidths = [0.15, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0]
orders = [0, 1, 2]

params = product(orders, bandwidths)
print("Bandwidth\tOrder\t\tRMSE")
print("------------------------------------------")
for o, b in params:
    lwr = LocallyWeightedRegression(b, o)
    lwr.fit(xtr, ytr)
    print(f"{b}\t\t{o}\t\t{-lwr.score(xts, yts):0.2f}")
    r_dict['Bandwidth'].append(b)
    r_dict['Order'].append(o)
    r_dict['RMSE (% bodyfat)'].append(f"{-lwr.score(xts, yts):0.2f}")

rdf = pd.DataFrame(r_dict)

with open("table.tab", "w") as f:
    f.write(rdf.to_latex(index=False))

# Validation error
# Best model is first order with bandwidth 0.5
lwr = LocallyWeightedRegression(bandwidth=0.5, order=1)
lwr.fit(xtr, ytr)
print(f"Best model val score: {lwr.score(xv, yv)}")
print(f"{lwr.get_params}")
```