Locally Weighted Regression

J.R. Powers-Luhn

***Abstract*—**

I. Introduction

II. Methodology

- Describe the locally weighted regression algorithm
- Discuss the pros and cons of locally weighted regression over least squares regression
- Discuss how weights are determined (Gaussian kernels)
- Discuss how the kernel bandwidth is optimized
- Discuss the importance of standardization for the weight calculation

A. Kernel regression

B. Locally weighted regression

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

The $\mathbf{W}$ matrix in equation 1 is a diagonal matrix 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 dataset of body measurements (body fat percentage, age, weight, height, adiposity index, and ten circumference measurements) was obtained [1]. Locally Weighted Regression was used to generate models to predict body fat percentage from the other predictor variables using cross validation to select the hyperparameter α.

III. Results

A. Model Selection

The model that was selected was that produced by the L-curve method. Its performance was not significantly different from the model produced using the LOO method and the larger α value should result in greater stability. Both models significantly improved on the least squares model, which had an RMSE of 19.35.

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

TABLE I: Lorem ipsum dolor simet

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

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%).

References

[1] 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}")
```