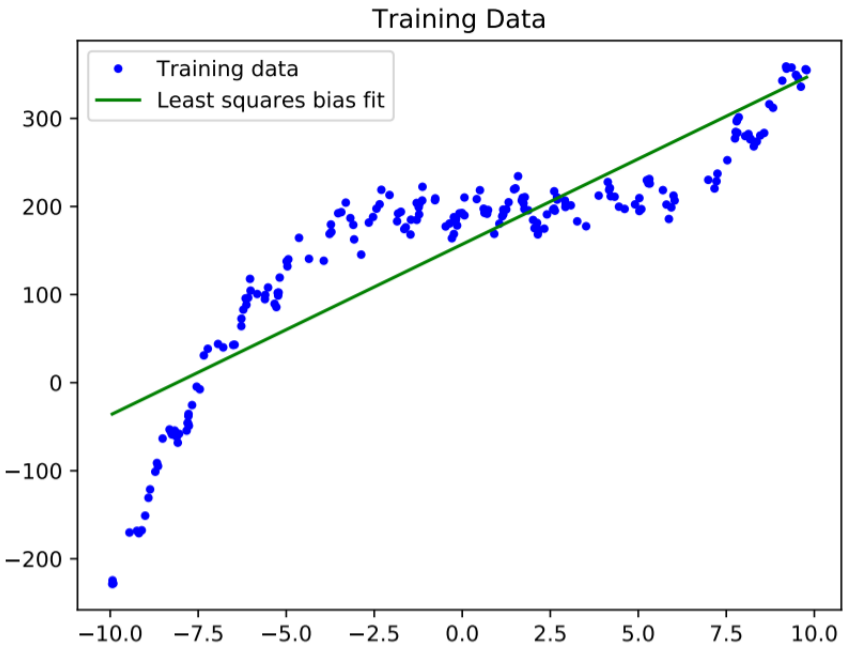
1 Linear Regression and Nonlinear Bases

Code: https://github.ubc.ca/cpsc340-2017S/sopida_zhenxil_a3/tree/master/code/linear_model.py

1.1 Adding a Bias Variable



https://github.ubc.ca/cpsc340-2017S/sopida_zhenxil_a3/blob/master/figs/Q1.1-leastSquaresBias.pdf

Training error = 3551.35

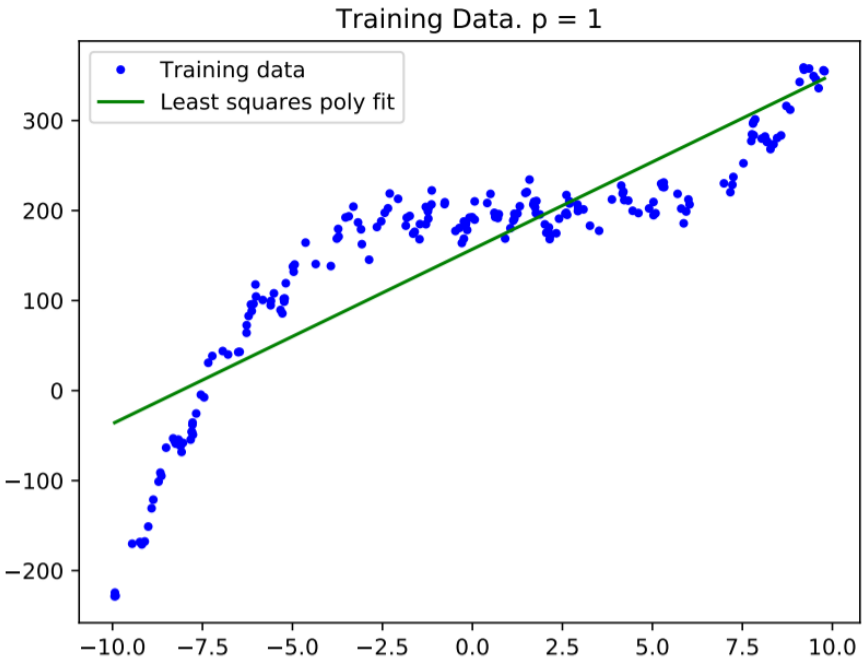
Test error = 3393.87

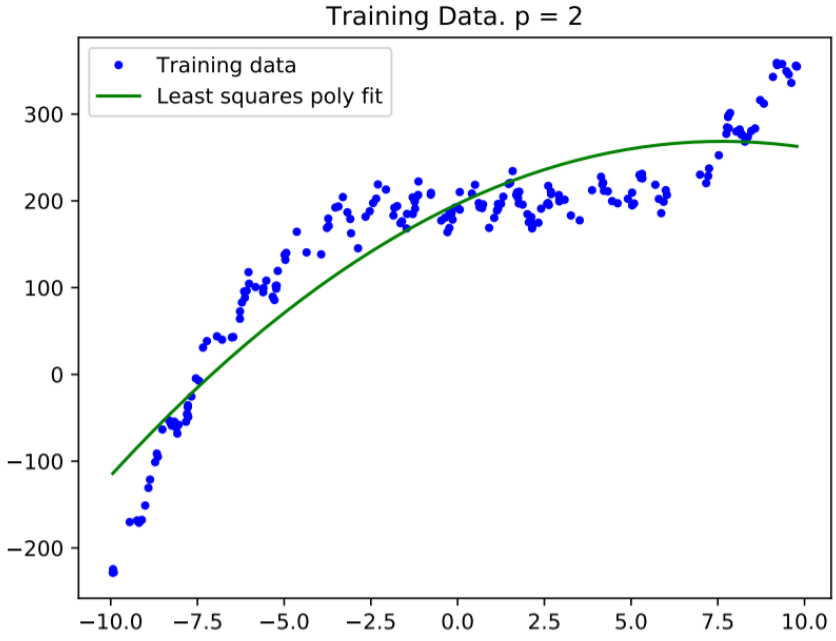
1.2 Polynomial Basis

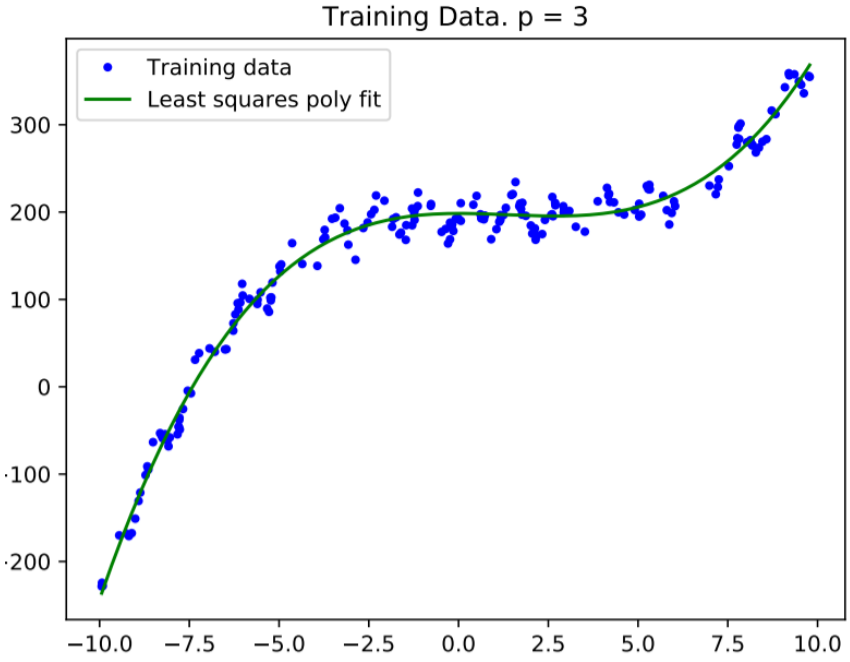
p=0
Training error = 15480.52
Test error = 14390.76
p=1
Training error = 3551.35
Test error = 3393.87
p=2
Training error = 2167.99
Test error = 2480.73
p=3
Training error = 252.05
Test error = 242.81
p=4
Training error = 251.46
Test error = 242.13

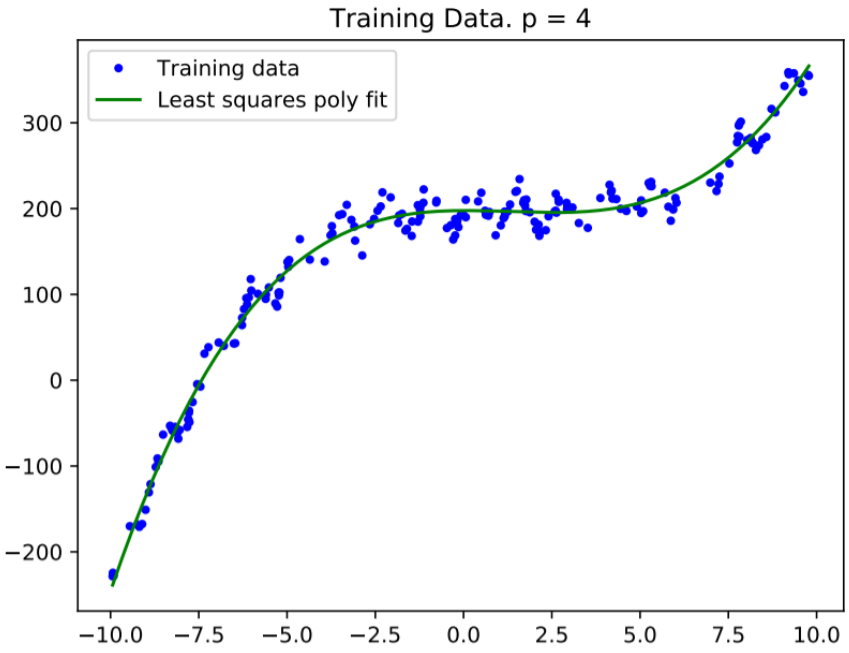
p=5
Training error = 251.14
Test error = 239.54
p=6
Training error = 248.58
Test error = 246.01
p=7
Training error = 247.01
Test error = 242.89
p=8
Training error = 241.31
Test error = 245.97
p=9
Training error = 235.76
Test error = 259.30
p=10
Training error = 235.07
Test error = 256.30

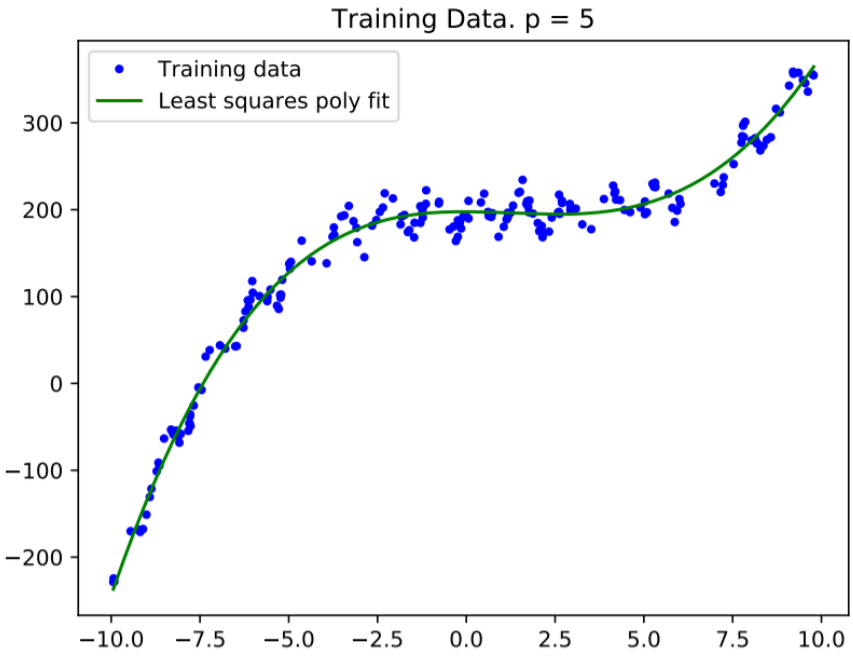
The training error decreases as value of p increases. However, the test error initially decreases but later increases as value of p increases. This implies that the model is starting to overfit.

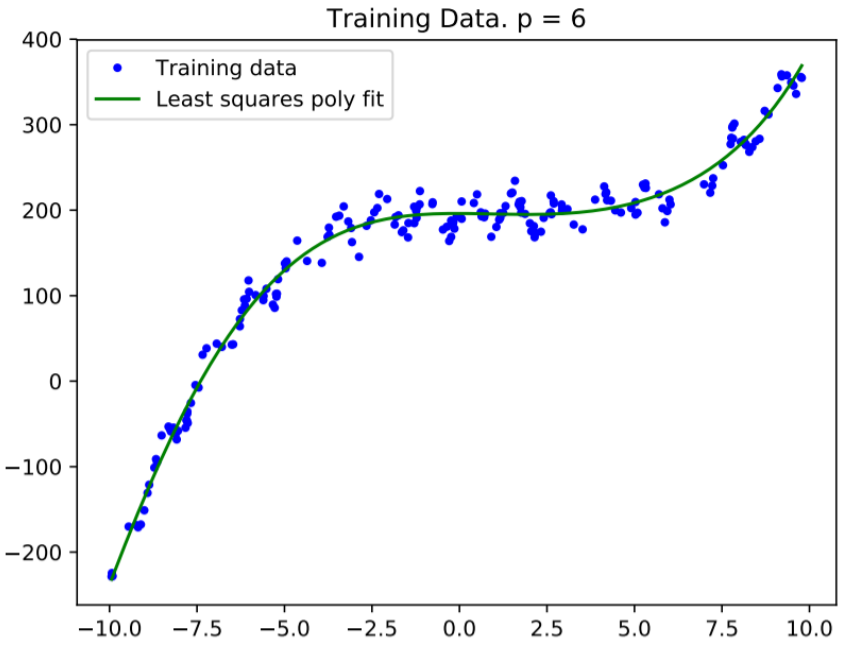


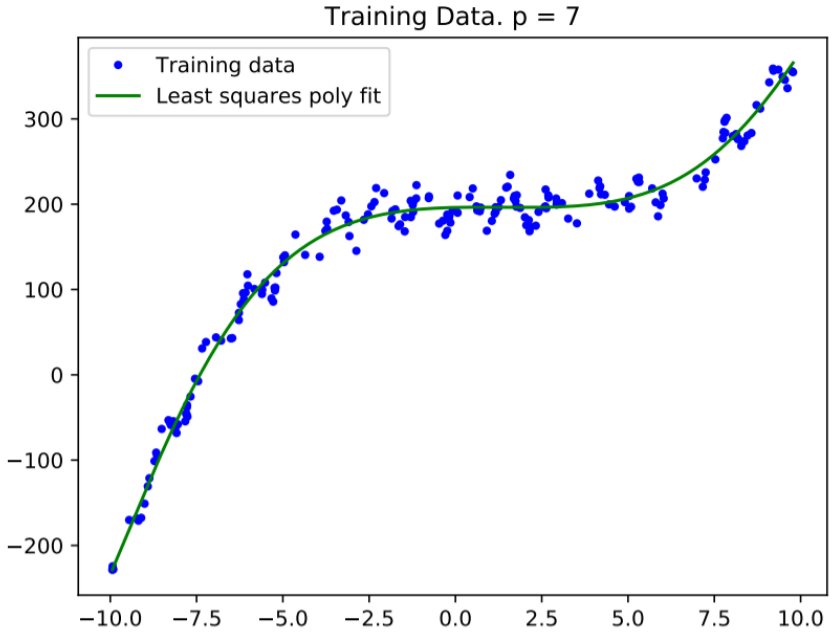

Training Data. p = 2
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

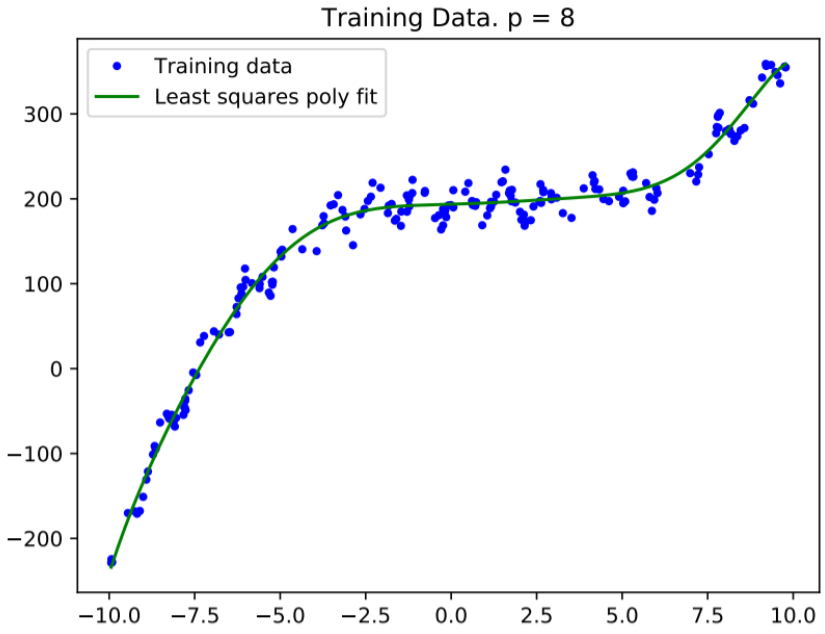

Training Data. p = 3
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

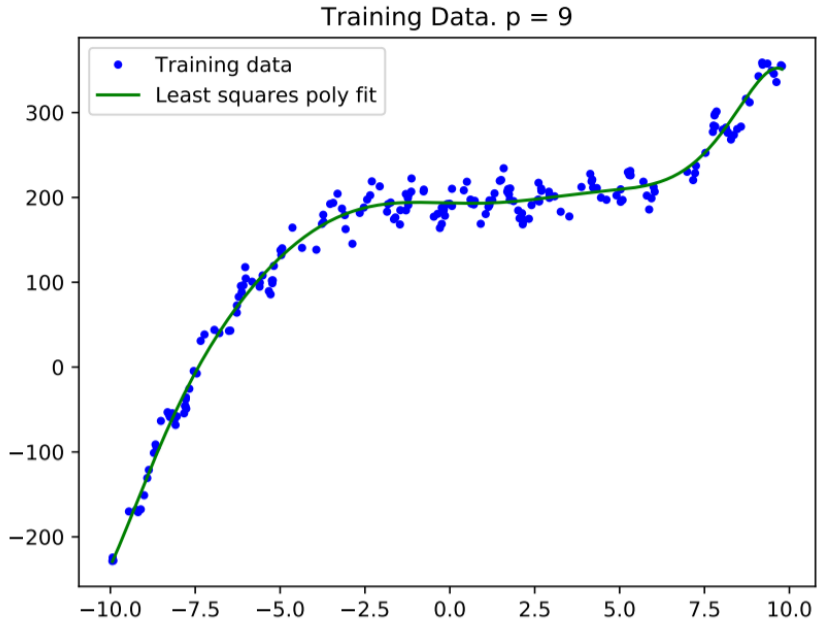

Training Data. p = 4
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

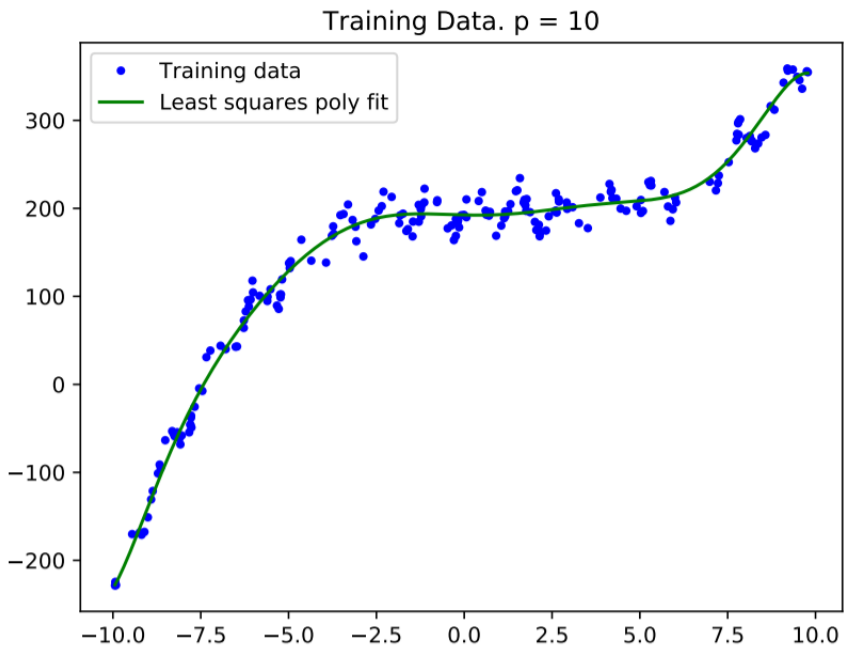

Training Data. p = 5
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Training Data. p = 6
Training data
Least squares poly fit
400
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Training Data. p = 7
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Training Data. p = 8
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Training Data. p = 9
Training data
Least squares poly fit
300
200
100
0
−100
−200
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



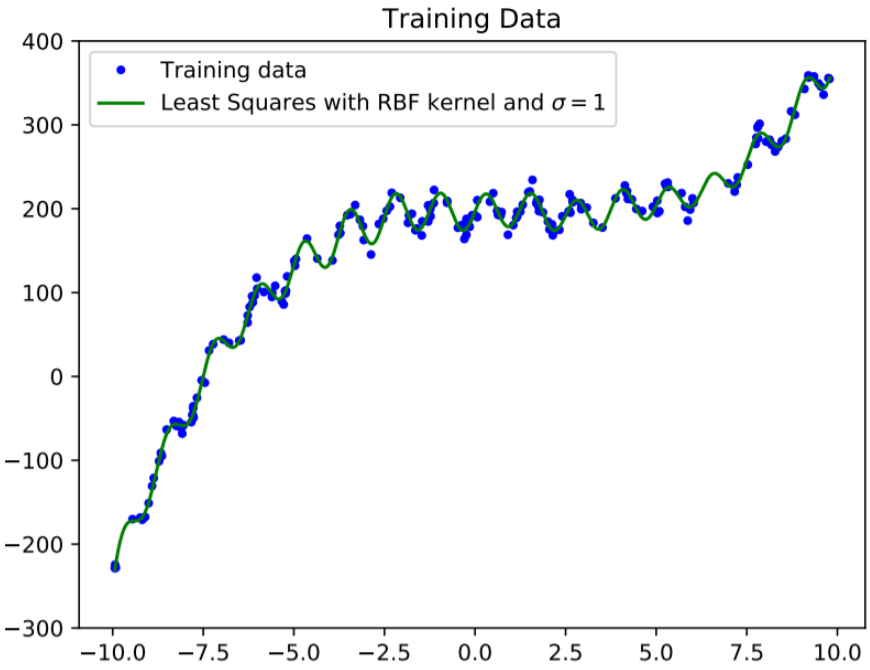
2 Non-Parametric Bases and Cross-Validation

2.1 Proper Training and Validation Sets

Since the training data (X) is sorted, the train and validation sets are not IID. To fix this problem, we can randomize the selection of training data for train and validation sets.

2.2 Cross-Validation

Code: https://github.ubc.ca/cpsc340-2017S/sopida_zhenxil_a3/tree/master/code/main.py



With 10-fold cross-validation, the procedure selects the best sigma value to be 1 with training error of 39.49 and test error of 71.17.

2.3 Cost of Non-Parametric Bases

(a) Linear basis

Training cost: X^TX costs $O(nd^2)$ to compute with $O(d^2)$ inner product between length n vector. For the inverse of X^TX, it costs $O(d^3)$. Hence, the total training cost is $O(nd^2 + n^3)$.

Prediction cost: each test example costs $O(d)$ to compute yhat. For t test examples, the prediction cost is $O(td)$.

(b) Guassian RBFs

Training cost: it takes $O(n^2)$ to construct Z (computing distances), and $O(n^3)$ to compute Z^TZ and inverse of Z^TZ. Hence, the cost of training is $O(n^3)$.

Prediction cost: it takes $O(tn)$ to construct $\hat{Z}$ (computing distances). $\hat{Z}w$ takes $O(tn)$ to compute. Hence, the prediction cost is $O(tn)$.

In Guassian RBFs, the number of features is n so it will be cheaper to train and test with RBFs when n < d.

3 Logistic Regression with Sparse Regularization

Code: https://github.ubc.ca/cpsc340-2017S/sopida_zhenxil_a3/tree/master/code/linear_model.py

3.1 L2-Regularization

logRegL2 Training error 0.002

logRegL2 Validation error 0.074

number of non-zeros: 101

3.2 L1-Regularization

logRegL1 Training error 0.000

logRegL1 Validation error 0.052

number of non-zeros: 71

3.3 L0-Regularization

Training error 0.000

Validation error 0.034

number of non-zeros: 24

3.4 Discussion

The number of non-zeros: L0 < L1 < L2. This means 'w' in L0-regularization have the most number of zeros i.e. closer to sparse vector. It also suggests that as 'w' gets closer to sparse vector, the validation error decreases. L0-regularization is more emphasizing on feature selection but the run time is slower.

4 Multi-Class Logistic

Code: https://github.ubc.ca/cpsc340-2017S/sopida_zhenxil_a3/tree/master/code/linear_model.py

4.1 One-vs-all Logistic Regression

logLinearClassifier Training error 0.084

logLinearClassifier Validation error 0.070

4.2 Softmax Classification

Class 1: $w_1^T \hat{x} = 2(1) - 1(1) = 1$

Class 2: $w_2^T \hat{x} = 2(1) + 2(1) = 4$

Class 3: $w_3^T \hat{x} = 3(1) - 1(1) = 2$

Since $w_2^T \hat{x}$ has the biggest value, the predicted class label for this test sample is 2.

4.3 Softmax Loss

$$\frac{d}{dW_{jc}}(f(W)) = (x_i)_j \left(-I(y_i = c) + \frac{\exp(w_c^T x_i)}{\sum_{c\prime=1}^{k} \exp(w_{c\prime}^T x_i)} \right)$$

$$= (x_i)_j (-I(y_i = c) + p(y_i | W, x_i))$$

4.4 Softmax Classifier

Training error 0.000

Validation error 0.008