Machine Learning HW1 – Linear Regression

NCTU EE 0310128 游騰杰

Programming language: Python

Environment: Python 2.7.13 under Homebrew @ Mac OS X 10.11.6

Python module used: numpy, pandas, math

Plot: Matlab

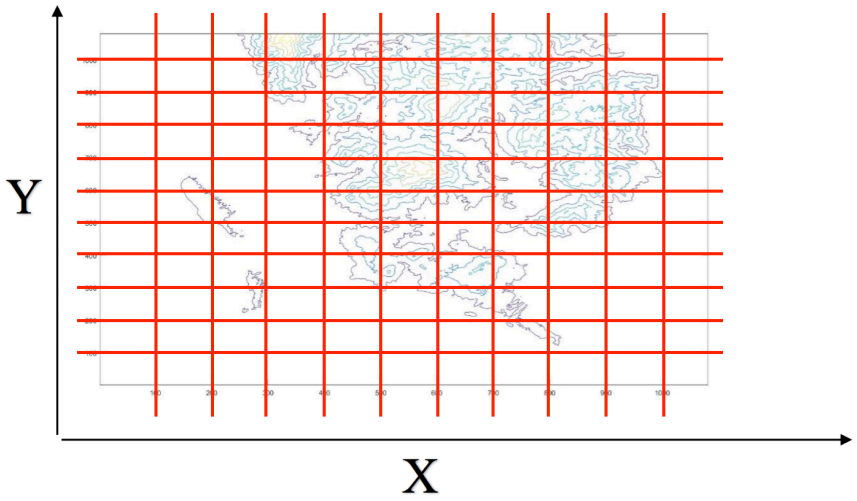
Overview:

Batch learning are used throughout this assignment.

All feature vectors are implemented with n^{th}-order 2-Dimension polynomials.

```
55
56  # Feature vector
57  def polynomial(x, y, order):
58      # Polynomial curver fitting
59      phi = np.array([0])
60      empty = True
61
62      for n in range(order + 1):
63          for x_n in range(n + 1):
64              if empty:
65                  phi = np.array([x**x_n * y**(n-x_n)])
66                  empty = False
67              else:
68                  phi = np.append(phi, x**x_n * y**(n-x_n))
69      return phi
70
```

Since the height of a map is a natural phenomenon, I divide the map into pieces (1296 in total,) so as to obtain a region (30 by 30 meters) whose variation in height is rather small, this feature is crucial when deciding the size of sub-regions, if the sub-regions are too small, there may not be enough data points for training or there’ s no data points inside the region; if the sub-regions are too big, the variation in height will be much bigger, which may require a more complicated model for a better result.



1. ML approach/MAP approach/Bayesian approach

A. Use ML, MAP, and Bayesian to construct 3 predictors to predict the heights, explain clearly why and how do you design your predictors.

(1) Maximum Likelihood (ML):

Formula [Reference] Textbook P.142 equation (3.15)

$$\mathbf{w}_{\mathrm{ML}} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

Implementation:

```
124 
125         # Regression Ch3 P.8
126         # Solve for w_ML
127         # w_ML = pinv(phi.' * phi) * phi.' * t
128         pinv = np.linalg.pinv(np.dot(dsgMtx.transpose(), dsgMtx))
129         w_ML.append(np.dot(np.dot(pinv, dsgMtx.transpose()), np.array(targt)))
130 
```

(2) Maximum A Posterior (MAP):

Formula: [Reference] Textbook P.145 equation (3.28)

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}.$$

Implementation:

```
 98         # Solve for w_MAP
 99         # w_MAP = pinv(phi.' * phi) * phi.' * t
100         pinv = np.linalg.pinv(regularizer * np.identity(dsgMtx.shape[1], dtype=np.int) \
101                               + np.dot(dsgMtx.transpose(), dsgMtx))
102         w_MAP.append(np.dot(np.dot(pinv, dsgMtx.transpose()), np.array(targt)))
103 
```

(3) Bayesian approach:

Formulas: [Reference] Textbook P.153 equation (3.50) (3.51),

$$\begin{aligned}\mathbf{m}_N &= \mathbf{S}_N\left(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}\right)\\ \mathbf{S}_N^{-1} &= \mathbf{S}_0^{-1} + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}.\end{aligned}$$

Formulas: [Reference] Textbook P.156 equation (3.57) (3.58) (3.59)

$$p(t|\mathbf{t},\alpha,\beta) = \int p(t|\mathbf{w},\beta)p(\mathbf{w}|\mathbf{t},\alpha,\beta)\,\mathrm{d}\mathbf{w}$$

$$p(t|\mathbf{x},\mathbf{t},\alpha,\beta) = \mathcal{N}(t|\mathbf{m}_N^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}),\sigma_N^2(\mathbf{x}))$$

$$\sigma_N^2(\mathbf{x}) = \frac{1}{\beta} + \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{S}_N\boldsymbol{\phi}(\mathbf{x}).$$

Implementation:

```
76
77  def bayesian(dsgMtx, beta, train_pts, t):
78      
79      mean = np.array(t).mean()
80      m_0 = np.full((np.shape(dsgMtx)[1], 1), mean)
81      tmp = []
82      empty = True
83      
84      S0 = np.identity(np.shape(dsgMtx)[1], dtype=np.int)
85      S0inv = np.linalg.pinv(S0)
86      #print np.shape(dsgMtx)
87      #print np.shape(S0inv)
88      #print np.shape(np.dot(dsgMtx.transpose(), dsgMtx))
89      SNinv = S0inv + beta * np.dot(dsgMtx.transpose(), dsgMtx)
90
91      #m_N = S_N * (S_0 inv * m_0  + beta * dsgMtx T * t)
92      S_N = np.linalg.pinv(SNinv)
93      m_N = np.dot(S_N, np.dot(S0inv, m_0).transpose()\
94                       + beta * np.dot(dsgMtx.transpose(), t))
95      return m_N[:, 0]
96
```

B. You may need to use some part of the training data as your testing data, and use those testing data to evaluate your methods.

After dividing the entire map into several sub-regions, first I observe the number of points per region, the mean sits around 30, so I decided to put a point into my verifying list whenever every six points are counted, which means roughly one sixth of the points in each region is used as testing data.

```
128             # Push verify data to the list
129             if itr_pts % skip_interval == 0:
130                 tmp_lst.append([pts_x, pts_y, subRegLst[itr_region][itr_pts][2]])
131                 continue
132             
144     
145     verify_lst.append(tmp_lst)
```

```
135     print "Verifying training result..."
136     arrayEmpty = True
137     
138     for itr_region in range(len(verify_lst)):
139         # w of the corresponding region
140         w_region = w_ML[itr_region]
141         for itr_pts in range(len(verify_lst[itr_region])):
142             # Acquire the test points
143             pts_x = verify_lst[itr_region][itr_pts][0]
144             pts_y = verify_lst[itr_region][itr_pts][1]
145             test = polynomial(pts_x, pts_y, order)
146     
147             # t_hat = w_T * x
148             targt_hat = np.dot(w_region.transpose(), test)
149             # Acquire the target
150             targt = verify_lst[itr_region][itr_pts][2]
151     
152             tmp_err = (targt_hat - targt) ** 2
```

Results:

(1) ML

```
$ python2 HW1_Polynomial_ML.py
Partitioning training data...
Calculating w_ML...
Verifying training result...
Making Predictions...
Exporting predictions...
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 2
  Number of Regions   = 1296
  Mean Square Error   = 176.991534261
  Standard Deviation  = 25.8905527577
  Maximum Error (abs) = 163.480343603
  Minimum Error (abs) = 0.0
================================================
```

(2) MAP

```
$ python2 HW1_Polynomial_MAP.py
Partitioning training data...
Calculating w_ML...
Verifying training result...
Making Predictions...
Exporting predictions...
=============Parameter Infomation==============
  Iteration : 1
  Lambda    : 10
================================================
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 2
  Mean Square Error   = 202.009050767
  Standard Deviation = 27.4221945883
  Maximum Error      = 163.399909533
  Minimum Error      = 0.0
================================================
```

(3) Bayesian

```
$ python2 HW1_Polynomial_Bayesian.py
Partitioning training data...
Calculating w_ML...
Verifying training result...
Making Predictions...
Exporting predictions...
===============Evaluation Result===============
  Sub-Region size     = 30
  Overlap             = 0
  Polynomial Order    = 2
  Number of Regions   = 1296
  Mean Square Error   = 177.593476175
  Standard Deviation  = 25.4253259608
  Maximum Error (abs) = 137.783195789
  Minimum Error (abs) = 0.0
================================================
```

C. Compare the difference and performance among the 3 methods.

I expected the MSE of MAP is smaller than that of ML with proper regularizer, but the result I obtained was quite the opposite. I tried all lambdas ranging from 0 to 10,000, the result gets worse monotonically. It is possible that the lambdas should be chosen specifically for each region due to the trade-offs between variation and bias may vary region by region. However, under the aforementioned situation, this method is computational costly when the amount of the sub-regions gets larger or the size of the sub-region gets smaller.

D. Try to make your predictor be under-fitting and overfitting for at least one methods and do some discussion

(1) ML

under-fitting

```
$ python2 HW1_Polynomial_ML.py
Partitioning training data...
Calculating w_ML...
Verifying training result...
Making Predictions...
Exporting predictions...
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 0
  Number of Regions   = 1296
  Mean Square Error   = 812.349290063
  Standard Deviation  = 48.9327277225
  Maximum Error (abs) = 244.321428571
  Minimum Error (abs) = 0.0
```

over-fitting

```
$ python2 HW1_Polynomial_ML.py
Partitioning training data...
Calculating w_ML...
HW1_Polynomial_ML.py:75: RuntimeWarning: overflow encountered
in long_scalars
  phi = np.append(phi, x**x_n * y**(n-x_n))
Verifying training result...
Making Predictions...
Exporting predictions...
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 10
  Number of Regions   = 1296
  Mean Square Error   = 3.29456616566e+46
  Standard Deviation  = 7.34967972407e+23
  Maximum Error (abs) = 4.52480110821e+24
```

(2) MAP

under-fitting

```
$ python2 HW1_Polynomial_MAP.py
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 0
  Lambda              = 10
  Mean Square Error   = 2372.59726401
  Standard Deviation = 80.4677908952
  Maximum Error      = 296.387096774
  Minimum Error      = 0.0
```

over-fitting

```
$ python2 HW1_Polynomial_MAP.py
HW1_Polynomial_MAP.py:39: RuntimeWarning: overflow encountered
 in long_scalars
  phi = np.append(phi, x**x_n * y**(n-x_n))
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 10
  Lambda              = 10
  Mean Square Error   = 3.29456616566e+46
  Standard Deviation = 7.34967972408e+23
  Maximum Error      = 4.52480110821e+24
  Minimum Error      = 0.0
```

(3) Bayesian

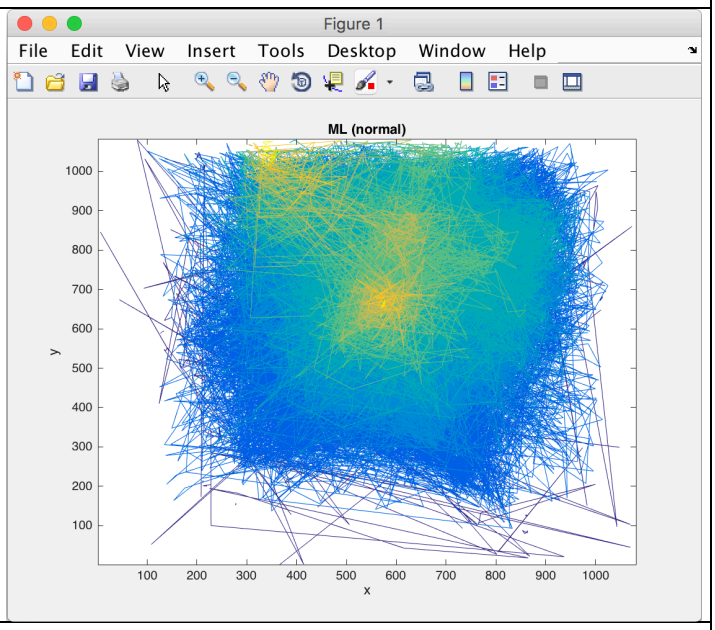
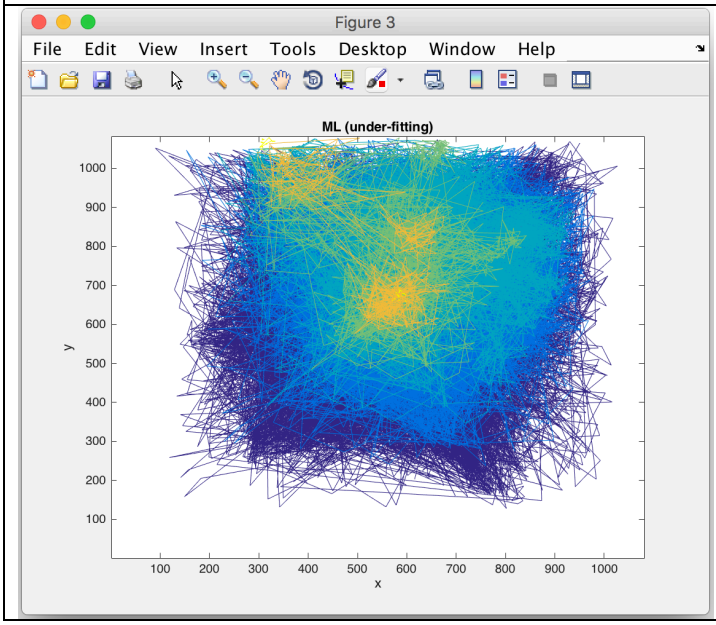
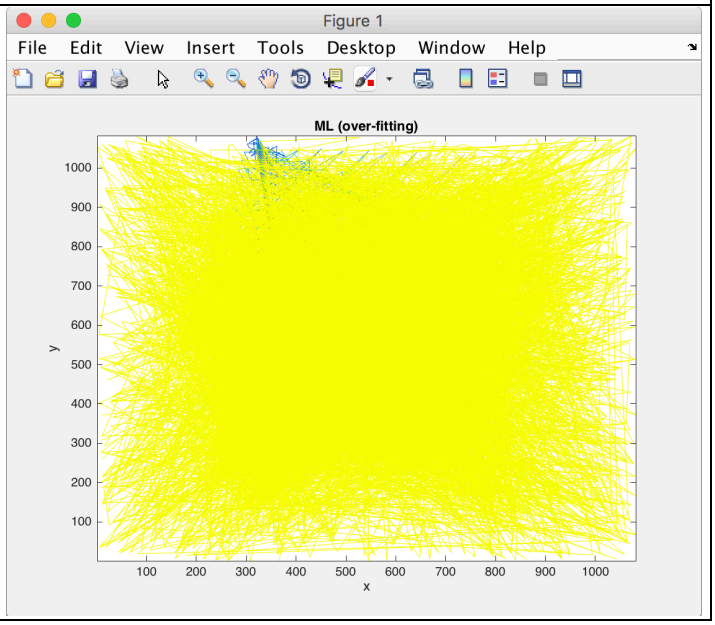
under-fitting

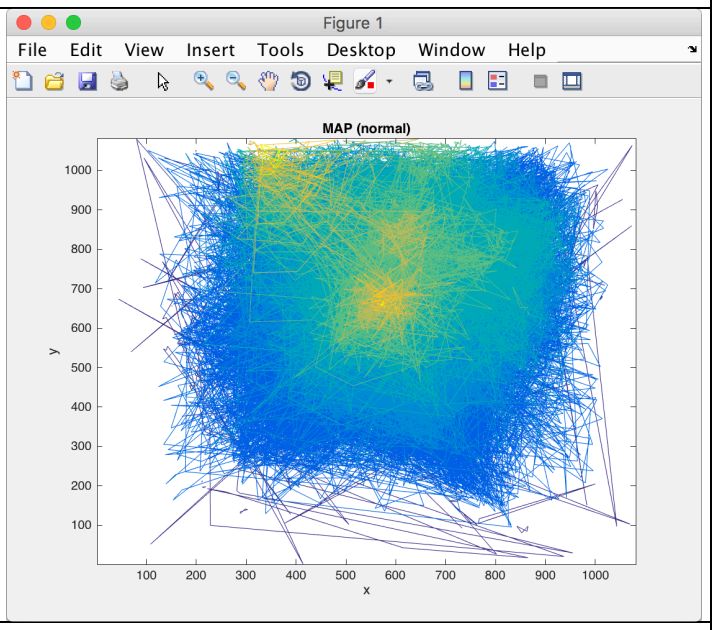
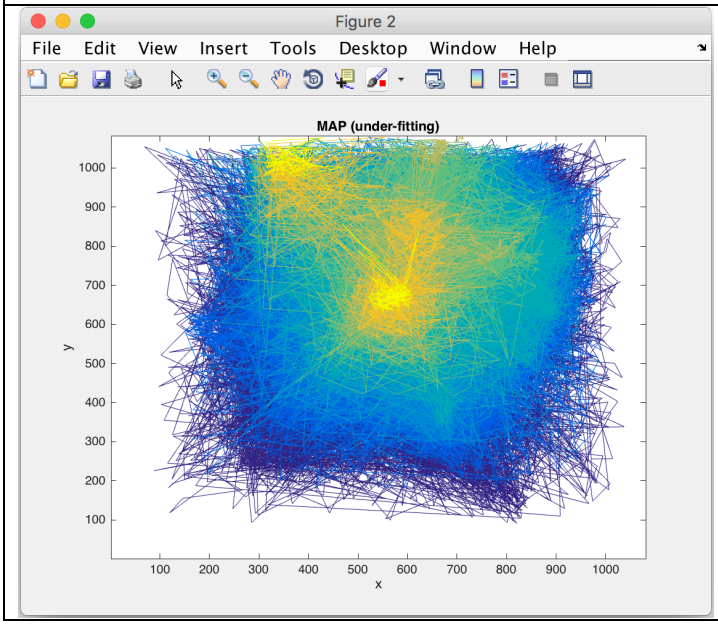
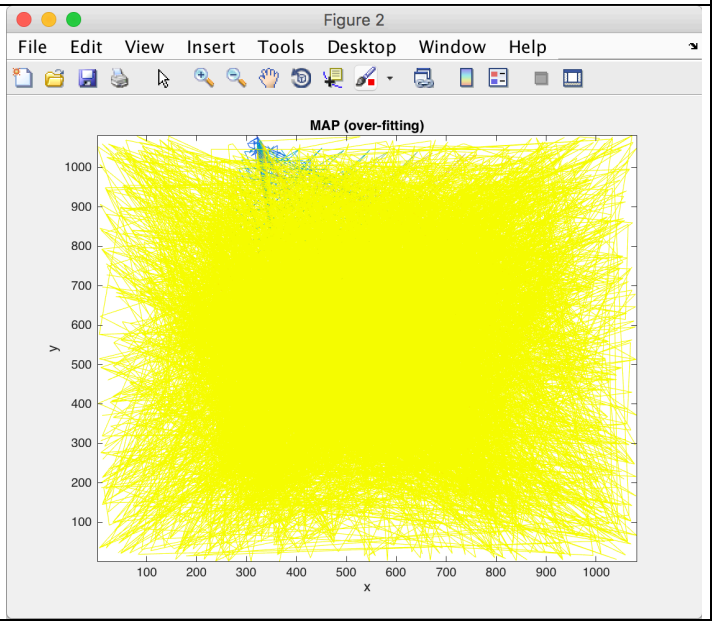
```
$ python2 HW1_Polynomial_Bayesian.py
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 0
  Number of Regions   = 1296
  Beta                = 0.0
  Mean Square Error   = 847.592052975
  Standard Deviation  = 48.7364896577
  Maximum Error (abs) = 232.103448276
  Minimum Error (abs) = 0.0
```

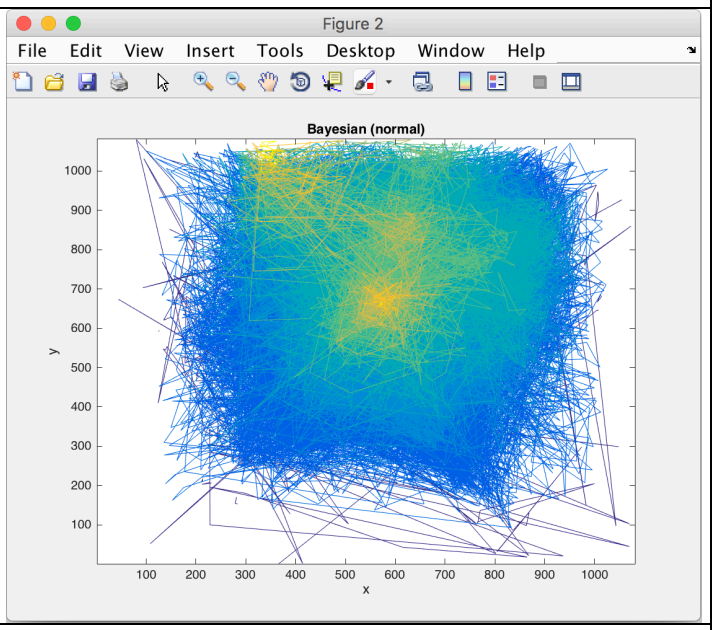
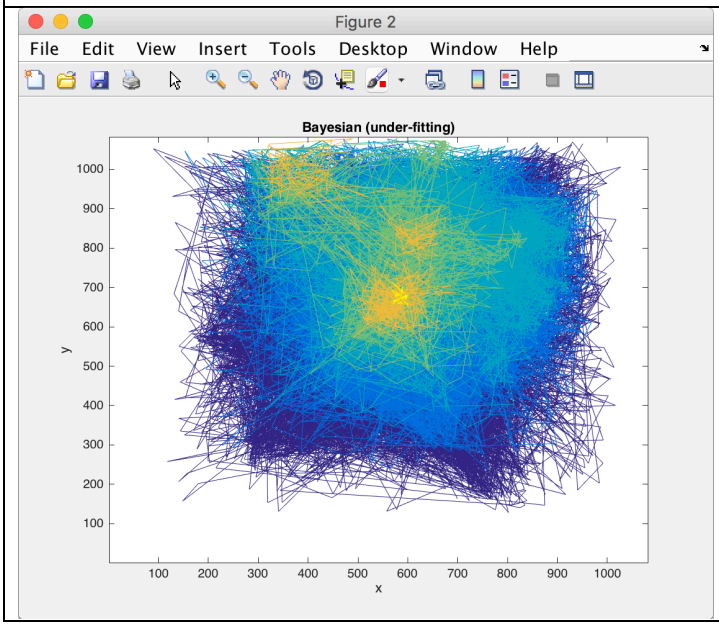
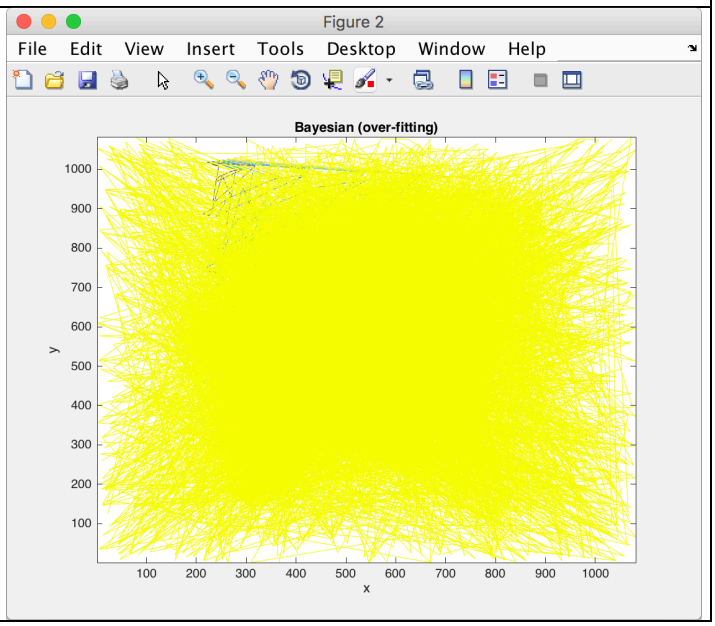
over-fitting

```
$ python2 HW1_Polynomial_Bayesian.py
===============Evaluation Result===============
  Sub-Region size     = 30
  Polynomial Order    = 0
  Number of Regions   = 1296
  Beta                = 0.0
  Mean Square Error   = 847.592052975
  Standard Deviation  = 48.7364896577
  Maximum Error (abs) = 232.103448276
  Minimum Error (abs) = 0.0
```

E. Visualize your height map will help you on the discussion

	normal
(1) ML	
under-fitting	over-fitting
	

	normal
(2) MAP	
under-fitting	over-fitting
	

	normal
(3) Bayesian	
under-fitting	over-fitting
	

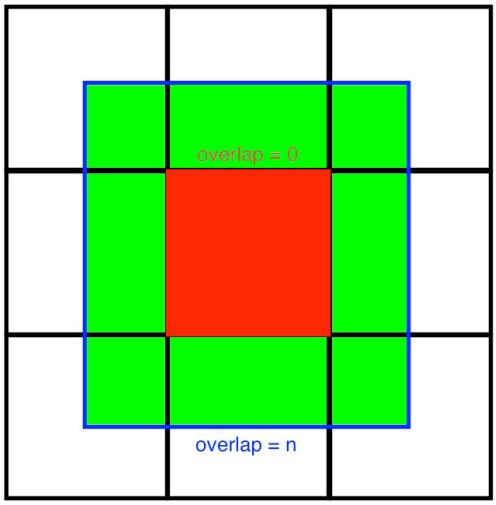
Discussion:

Generally, the under-fitting makes the variation in height higher compared to the normal one, the height become less continuous, several medium values are not properly predicted. On the other hand, overfitting generally made the height become indistinguishable.

F. Other discussion may help you to get higher score in this Homework

- Overlapping of the partitioned sub-regions:

This method is motivated by Moving Average I learned in Signals and Systems, which incorporates lateral points of each region as training data so as to improve the prediction by means of considering nearby points for continuity of the height.



As illustrated on the right, normally, when the overlap is set to zero, we only considers each region inside the red region. If the overlap is set to other nonzero positive values, the region we trained gradually expands as the green region shows. This way, we can be able to consider lateral changes around the region of interest, hoping this will improve the accuracy of our model. The implementation is shown below,

```
print "Partitioning training data..."
for x_rng in xrange(region_Size, max_X + 20, region_Size):
    for y_rng in xrange(region_Size, max_Y + 20, region_Size):
        # initialize the np.array
        subRegion = np.zeros(shape=(1,3), dtype=np.int)
        arrayEmpty = True

        entry = 0
        for itr in range(num_Data):
            if (rawX[itr] <= x_rng + overlap) and \
            (rawX[itr] >= x_rng - region_Size - overlap) and \
            (rawY[itr] <= y_rng + overlap) and \
            (rawY[itr] >= y_rng - region_Size - overlap):

                tmp_array = np.array([rawX[itr], rawY[itr], rawT[itr]])

                if arrayEmpty:
                    subRegion = tmp_array
                    arrayEmpty = False
                else:
                    subRegion = np.vstack([subRegion, tmp_array])
                entry = entry + 1

        # Push the valid number of points in each region to the list
        valid_list.append(entry - 1)
        subRegLst.append(subRegion)
        # move to next region
        region = region + 1
```

But it came out that the result did not meet my expectation, the MSE became larger, I guessed it may result from the raw data we used has already been pre-processed, many information and features has been discarded, including the local continuity of the target value, so properties that can be applied to the ground truth data vanish. Therefore, I set the overlap to zero in the code, but the parameter can still be used.

2. Cross validation (N = 3, order = 2)

Lambda	Iteration	MSE	Standard Deviation
0	1/3	209.32	30.36
0	2/3	202.57	27.55
0	3/3	207.16	29.77
100	1/3	225.60	30.60
100	2/3	216.59	27.80
100	3/3	233.76	30.88
200	1/3	228.25	30.60
200	2/3	216.03	27.42
200	3/3	233.72	30.49
300	1/3	232.24	30.70
300	2/3	217.66	27.31
300	3/3	235.52	30.26
400	1/3	236.99	30.86
400	2/3	220.61	27.34
400	3/3	238.49	30.14
500	1/3	242.16	31.05
500	2/3	224.34	27.44
500	3/3	242.21	30.10

Method: MAP
Hyperparameter: Lambda

Discussion:
Cross validation is used identify how well the learning parameters generalize across the samples we learn from in each fold. Here, the second part of the data gives lower MSE while other two parts of data give similar results. With larger lambda, the MSE monotonically increases.

[Reference: http://stats.stackexchange.com/questions/43131/cross-validation-and-parameter-optimization]