CS584: Assignment 1

Pooja Mankani
Department of Computer Science
Illinois Institute of Technology
02/21/2016

Abstract

The report illustrates the implementation of parametric regression on synthetic and real data sets. The 10 fold cross validation for all the implementations is considered and the training error and testing error is evaluated. A custom model is created and the corresponding python built in model is also created using sklearn package in python. The errors of each of the model is compared and analyzed.

1. **Problem Statement**

   The following problems are considered in the implementation:

   a. **Single variable regression:**
      - Linear Regression on a single variable dataset: This implementation can be used for datasets that are dependent on a single feature and can be predicted only using that feature alone. For e.g. Stocks sold by a company over a period of time
      - Polynomial Regression on a single variable dataset: This implementation can be used for datasets where the degree of the single feature is increased in order to get multiple variations of a single feature. For e.g. predicting the length of the blue fish based on its age. Since the length of the fish and age is a quadratic plot the polynomial regression model must be used.

   b. **Multiple variable regression:**
      - Load multiple feature data sets into a higher dimensional space: This is required since multiple features in a lower dimensional space are retained as polynomial however, in a higher dimensional space they become linear due to multiple features which makes it easy to solve the regression problem
      - Linear Regression in a higher dimensional space: This implementation can be used for data sets that have limited multiple features. In this case the features can be increased in a higher dimensional space and then values can be predicted. For e.g. the polio data set where the crucial features are if the child is vaccinated or not and if the child contracted to polio within a certain period of time.
      - Solve regression using iterative solution: This implementation is important since the model may have errors and would not learn as required. Hence we iteratively improve the model until the time we get the least error and maximum accuracy
      - Gaussian Kernel function: This implementation is important for data sets where the number of samples are less that the number of features.

2. **Proposed solution**

   a. **Single variable regression:**
      - Linear Regression using single variable:
         i. A linear model is created using the number of samples, the sum of the features and its squares and the sum of its corresponding target values.
         ii. This model is then used to predict the testing data. The training and the testing errors are compared.

- **Polynomial Regression using single variable:**
    i. A polynomial model is created such that the feature is raised to the power of all the degrees. This feature matrix is then used to predict the target values. For e.g.: if the degree is 3 then the feature matrix would have 4 columns as 1, X1, X1^2, X1^3
    ii. This polynomial model is then used to predict testing data and a comparison is made between the training and testing errors.
    iii. Various degrees of the polynomial is tried and the one that provides the best accuracy with least bias and variance is considered.

b. Multi Variable Regression:

- **Multiple feature in higher dimensional space:**
    i. The features in the data set are mapped to a higher dimensional space using the sklearn.preprocessing.PolynomialFeatures. This library helps to create the multiple combination of features based on the original number of features and the degree.
    ii. Once the features matrix is created with polynomial features it is mapped to a higher dimensional space.
- **Linear Regression in higher dimensional space:**
    i. A linear model is created on the multiple combinations of feature matrix. This model is created with a dot product of pseudo inverse of the feature matrix with target training data.
    ii. Once the model is created it is applied on the testing data and the error is calculated. The error is compared with the training error.
    iii. A degree which creates a most accurate model will be selected.
- **Regression using iterative solution:**
    i. The model implemented in the previous step is an explicit solution. An iterative solution is also implemented for the various data sets.
    ii. The iterative solution is implemented for 1000 iterations with a learning rate of 0.01.
    iii. The feature matrix is mapped to a higher dimensional space. Initially the model is considered to be have all ones. With a combination of the feature matrix and the initial model training target values are predicted. The difference between the predicted values and the actual values are checked and based on the difference the model is evaluated and improved iteratively. In order to calculate the model the difference is divided by the number of samples in order to normalize the data.
    iv. This model is then used to predict the testing data and the errors are compared.
- **Gaussian Kernel function:**
    i. The kernel function is applied on the feature matrix. The feature matrix is mapped to a higher dimensional space. Using this matrix the similarity between all the vectors in the matrix is calculated which is denoted as a Gram Matrix.
    ii. This matrix is then used to evaluate alpha. Using a combination of alpha and the similarity between the testing input data and training data the testing target data is evaluated.
    iii. The sigma value is used to predict the testing data. For different data sets the value of sigma would be different based on the accuracy.

c. **Comparison and Analysis of results:**
    - Training and Testing error: The training and testing error is evaluated for each of the models and the same is compared and analyzed.
    - Error of custom and python model: The testing error of the custom model created in this assignment and the python model using sklearn is compared and analyzed.
    - Error of explicit and iterative solution: The testing error of the explicit solution is compared and analyzed with the iterative solution (gradient descent)
    - Error and performance of Gaussian kernel and linear regression: The testing error of Gaussian kernel and linear regression is compared using the results and the time taken by each of them.

3. Implementation Details:

a. **Design issues:** The following design issues were witnessed:
    - The polynomial regression model for multiple feature is difficult to plot and cannot be plotted in a two dimensional space since there are multiple features associated to it with a quadratic function.
    - The matrix for polynomial with respect to higher degree is difficult to obtain manually. This is complex since the combination of features is difficult to construct along with the order of the same. For e.g. if the degree is 2 and the number of features is 2 then the various feature combinations would be: **[1 X1 X2 X1^2 X2^2 X1X2].** Evaluating these combinations is difficult and complex.
    - In case multiple records for large data sets are given as a input, for e.g. consider a data with billion records, in this case if we store all the data extracted from the records into variable then there could be a possibility of memory error within your system. This is because it would take up lots of memory to store data within systems.

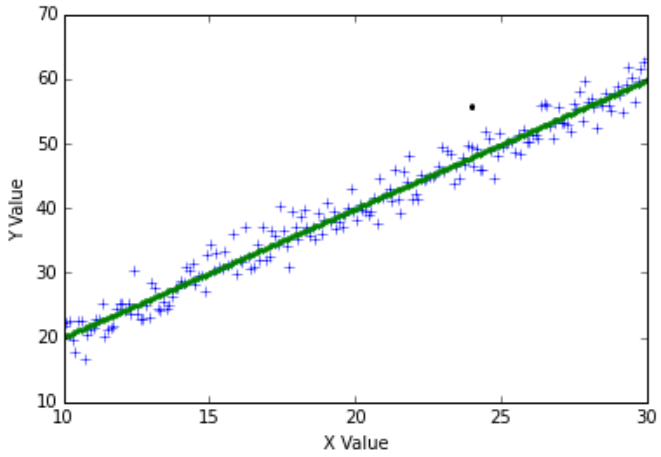
b. **Problems faced:** The following issues were faced:
    - Cross validation: The implementation of cross validation to retrieve the training and testing indices was complex to obtain using python array and matrix manipulation. Hence in this case KFold cross validation of sklearn package was used to evaluate the same.
    - Features in higher dimensional space: The multiple combination of features in the higher dimensional space is complex to evaluate using python functions. Since for different degrees and different number of features the combinations become complex. Hence in this case PolynomialFeatures of the sklearn.preprocessing package is used to evaluate the same.
    - Similarity between two vectors for Gaussian kernel: The similarity between two vectors for the kernel method was difficult to implement since the Euclidian distance of the vectors need to be calculated. Hence in this case the np.linalg.norm method was used to get the distance and hence the similarity between the vectors.
    - Sigma value for Gaussian kernel: The sigma value was difficult to evaluate since there is no default value for sigma. It depends on different data sets. Hence we need to perform trial and error for the sigma value to get the maximum accuracy.
    - Gradient: The model evaluated by gradient were resulting in high errors hence the data was required to be normalized. Thus the formula was updated such that the difference between the actual and the predicted value was divided by the number of samples.

c. **Implementation Details:**

Python Notebook	Command	Example
Assignment_1_Linear_Single.ipynb	run_linear_regression_single(filename,num_folds)	run_linear_regression_single("svar-set1.dat", 10)
Assignment_1_Poly_Single.ipynb	run_poly_regression_single(filename,num_folds)	run_poly_regression_single("svar-set1.dat", 10)
Assignment_1_Poly_Multi.ipynb	degree=2 input_data=read_input("mvar-set4.dat") data_x=create_feature_matrix(input_data) data_y=create_y_matrix(input_data) avg_custom, avg_python=do_cross_validation(data_x, data_y,degree, 10) print "(Custom & Python Model) Average RMS: %.15f, %.15f" %(avg_custom, avg_python) theta=poly_model_multiple(data_x, data_y, degree) predicted_y=predict(data_x, theta, degree);	
Assignment_1_Gradient.ipynb	run_gradient_iterative(filename,fold,degree,learning_rate)	run_gradient_iterative("mvar-set1.dat", 10, 3, 0.01)
Assignment_1_Gausian_Kernel.ipynb	run_gaussian_kernel(filename, sigma,fold)	run_gaussian_kernel("mvar-set1.dat", 1, 10)

4. Results and Discussions:
   a. Single variable regression:
   - **Linear Regression: (1a, 1b, 1c)**
     - **Single Variable Data Set 1: (10 fold validation):**

Model overfits the data



Training error has no significance compared to testing error

Training Error	31.601423
Testing Error	3.149564

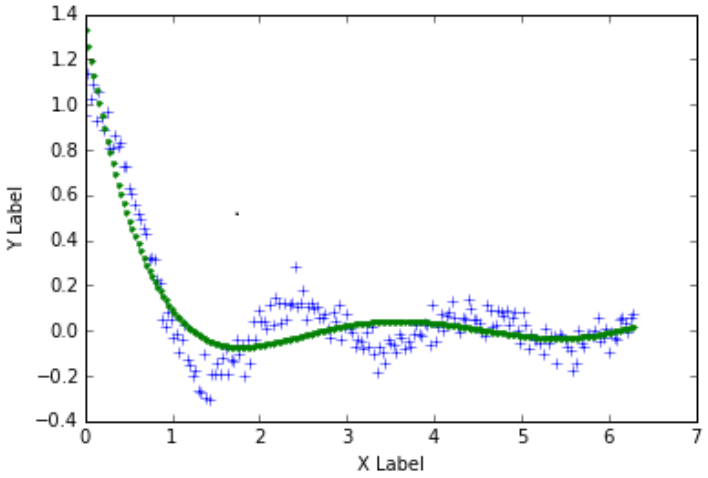
Custom model performs exactly well as the python built-in model

Custom Model Error	3.149564
Python Built in Error	3.149564

The other data sets did not fit well on the linear model. The details can be seen in the python notebook (Assignment_1_Linear_Single)

- **Polynomial Regression: (1d)**
  - **Single Variable Data Set 2: (10 fold validation): Model fits well at degree=5; at degree 6 it starts overfitting**

Degree = 5 (Data Set 2) (Model fits well)



Comparison of errors at different degrees. At degree=5 model works well but at 6 overfitting occurs since variance increases and bias decreases.

Degree	Data Set 2	Data Set 3
	Testing Error	Testing Error
2	34.305056	279.337201
3	26.85462	270.761637
4	17.744521	283.013882
5	11.797786	276.735039
6	13.138989	1081.133427

Training error has not significance compared to testing

Training Error	361.998494
Testing Error	11.797786

Custom model performs as well as python model

Custom Model Error	11.797786
Python Built in Error	11.797786

Other data sets can be checked in the python notebook (Assignment_1_Poly_Single)

Effect of performance for linear and polynomial models as the dataset reduces: (1e)

- The dataset reduction is performed by reducing the folds in the cross validation. As the number of folds reduce the training error and the testing error change since the partitions change.
- Since the partition are randomly generated there is no specific pattern of the reduction
- However, the performance of the model increase with reduction in dataset since the execution time is less for smaller datasets

b. Multiple variable regression:

- **Polynomial regression (2a, 2b)**

The model fits well for degree = 3 greater than the degree it overfits and lesser than the degree it underfits

Degree	Data Set 1	Data Set 2	Data Set 3
	Testing Error	Testing Error	Testing Error
2	0.25961663	0.020037596	0.250836335
3	0.2606603	0.010402279	0.010402279
4	0.26030548	0.010439268	0.251115459

Custom model error and python model error have the exact results

Custom Model Error	0.01040228
Python Built in Error	0.01040228

Additional details can be seen in the notebook (Assignment_1_Poly_multiple)

- **Regression using iterative solution (Assignment_1_Gradient): (2c)**

The iterative solution works well when the learning rate is 0.01 however, in case we reduce the learning rate even further the errors increase. This is because we are limiting the number of iterations to 1000 and we are asking the model to learn to a lesser extent.

Values with learning rate = 0.01

Degree	Data Set 1	Data Set 2
	Testing Error	Testing Error
2	0.260086	0.020337
3	0.26278	0.013089
4	0.279274	0.015667

Values with learning rate = 0.001

Degree	Data Set 1	Data Set 2
	Testing Error	Testing Error
2	0.343919	0.253506
3	0.392246	0.138204
4	0.439232	0.369397

Training error and testing error are almost similar (Data set 1)

Training Error	0.259735
Testing Error	0.26278

Training error and testing error are almost similar (Data set 2)

Training Error	0.012943
Testing Error	0.013089

Other data sets can be checked in the python notebook (Assignment_1_Gradient)

- **Gaussian Kernel (Assignment_1_Gaussian): (2d)**
  The error rate for Gaussian Kernel with sigma = 1 is 0.328849

The accuracy of the model is received best when the sigma value = 1. For other detail execution Assignment_1_Gaussian notebook can be checked

Time Taken (seconds)	Model
29.75163909	Poly Multiple Variable
5012.247656563056807	Gaussian Kernel

The time taken by Gaussian Kernel is more than the primal method. This is because the similarity between each of the vectors is calculated which takes time. Moreover, Gaussian kernel should be solved only for dual degree problems where number of samples is less than the number of features. Hence Gaussian kernel method would not work well for these data sets

References:

http://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/

http://www.math.uah.edu/stat/sample/Introduction.html

http://www.gaussianprocess.org/gpml/chapters/RW2.pdf

http://stackoverflow.com/questions/17784587/gradient-descent-using-python-and-numpy

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html