CS F320: Foundations of Data Science
Assignment 2 Report

Siddarth Gopalakrishnan - 2017B3A71379H
Rohan Maheshwari - 2017B4A70965H
Satvik Vuppala - 2017B4A71449H

Contents

Data Description

```
In [53]: dataset.describe()
```

Out[53]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

where age, bmi and number of children are the feature variables and charges is the target attribute which we have to predict.
There are a total of 1338 data points.

Pre Processing

Variables that are measured at different scales do not contribute equally to the model fitting & learned function and might end up creating a bias. We can clearly see from the above data description that min and max values of the features differ widely. Hence to solve this problem we standardized all the features of the dataset to have mean 0 and unit variance (μ=0, σ=1).
We used StandardScaler model from sklearn package for the same.

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Models

We used 3 models for finding the best fit line for the given data:

1. ### Solving Normal Equation

   It simply refers to finding the inverse of the design matrix to solve the regression problem.

   Our model

   $$X\mathbf{w} \approx \mathbf{y}$$

   We have to minimize

   $$\mathbf{e} = \mathbf{y} - X\mathbf{w}$$

   we will take the square of this error to be our objective function:

   $$J(\mathbf{w}) = \|\mathbf{e}\|^2 = \|\mathbf{y} - X\mathbf{w}\|^2$$

   Solving:

   - $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} J(\mathbf{w}) = \arg\max_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|^2$
   - let's expand $J(\mathbf{w})$:
     - $J(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2 = (\mathbf{y} - X\mathbf{w})^T(\mathbf{y} - X\mathbf{w}) = \mathbf{y}^T\mathbf{y} - (X\mathbf{w})^T\mathbf{y} - \mathbf{y}^T(X\mathbf{w}) + (X\mathbf{w})^T(X\mathbf{w}) = \dots$
     - $\dots = \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T X^T\mathbf{y} + \mathbf{w}^T X^T X\mathbf{w}$
   - now minimize $J(\mathbf{w})$ w.r.t. $\mathbf{w}$:
     - $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -2X^T\mathbf{y} + 2X^T X\mathbf{w} \stackrel{!}{=} \mathbf{0}$
     - $X^T X\mathbf{w} = X^T\mathbf{y}$ or
   - the solution:
   - $\mathbf{w} = (X^T X)^{-1} X^T\mathbf{y} = X^+\mathbf{y}$
   - where $X^+ = (X^T X)^{-1} X^T$ is the Pseudoinverse of X

   Hence, we simply have to solve:

   $$\mathbf{w} = (X^T X)^{-1} X^T\mathbf{y}$$

2. Gradient Descent Algorithm

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

We have the line equation to be approximated as:

*Charges = w0 + w1*age + w2*bmi + w3*children*

So we need to find w0, w1, w2, w3

We will use RMSE error function which is:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Representing in vector format we get:

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \ldots + \theta_j x_j^{(i)}$$

Error function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Derivative:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Parameter Update step:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

We can vectorize the cost function as:

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

The derivation of cost function to all θ can be vectorized as:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T(X\theta - y)$$

Hyperparameters: (after tuning)
alpha = 0.0001
epochs = 200000

3. Stochastic Gradient Descent

Similar to gradient descent but instead of going through all the training data in each iteration, we select a random data point in each iteration and use that to update our parameters.
We shuffle the data points before applying SGD every time to obtain a less biased estimation of the true gradient.

Hyperparameters: (after tuning)
alpha = 0.01
max_epochs = 1,000,000

OBSERVATIONS

Took 20 random 70:30 splits of data set into training and testing data to build 20 regression models to compile the following observations.

Training and Testing Errors(rmse)

	TRAINING		TESTING	
	Min	Max	Min	Max
Normal Equations	0.9148518666171795	0.9710127050984874	0.8624792256008518	0.9920312156089844
Gradient Descent	0.9148518666172893	0.9710127050986988	0.8624792256840245	0.9920312155256973
Stochastic Gradient Descent	0.9148518715694625	0.9710127452085308	0.8624567586387246	0.9920235208966341

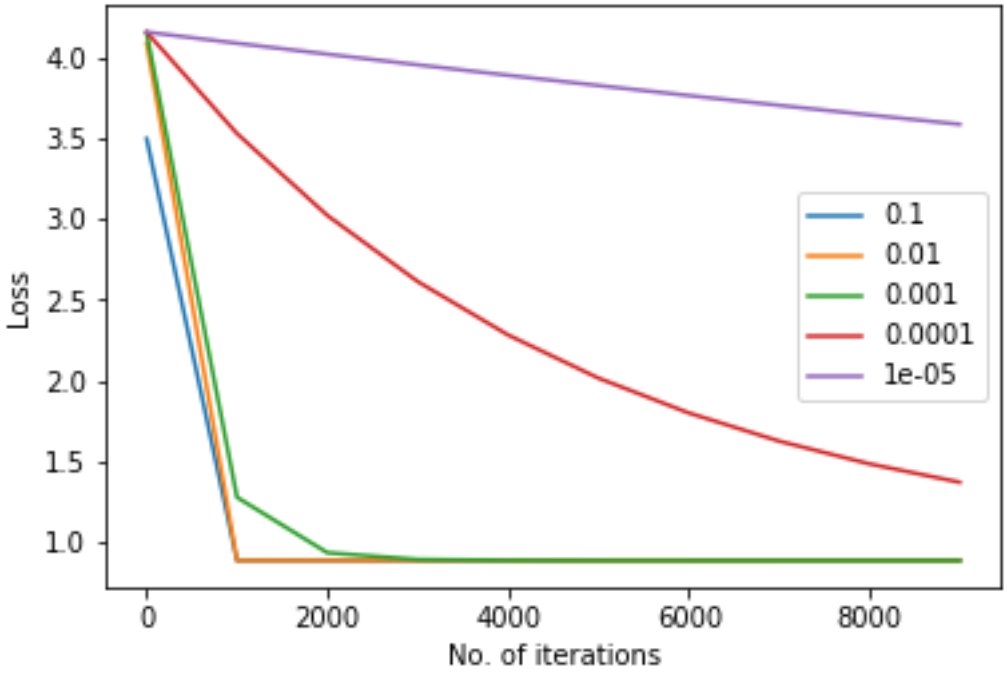
Mean and Variance of prediction accuracy(rmse)

	TRAINING		TESTING	
	Mean	Variance	Mean	Variance
Normal Equations	0.9411132902691401	0.000158067705628911	0.9317115649610962	0.000158067705628911
Gradient Descent	0.9411132902692001	0.000158067705629359	0.9317115650326968	0.000863995723471830
Stochastic Gradient Descent	0.9411133080503271	0.000158067820170784	0.9317103985070535	0.000864136248928941

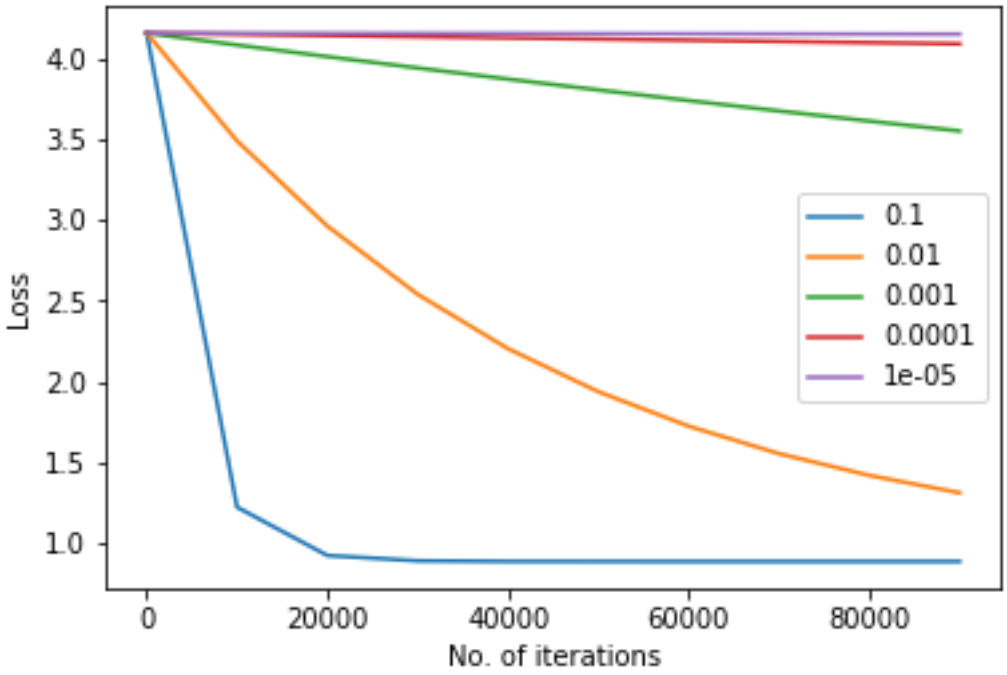
We can clearly see that all 3 models give almost similar results. Comparatively normal equations give minutely better results because it is an analytical approach whereas the other 2 are numerical approaches. Similar results of the 3 models could also be attributed to the small size of the dataset (1338 data points).

ERROR VS EPOCHS

- Gradient Descent



- Stochastic Gradient Descent



(Loss here represents Mean Squared Error)

RESULTS

- *Comparing the 3 models*

  Looking at mean and variance results of the 3 models we can say that all 3 give almost the same results. It might not happen all the time but here due to small dataset size and proper hyperparameter tuning we have managed to achieve optimal results in all 3 models.
  Most efficient method out of the 3 for real world applications would definitely be Stochastic Gradient Descent as the other 2 models have major flaws once our dataset gets bigger and bigger.
  For normal equations computing large matrix inverses will be quite time and memory intensive. As for gradient descent iterating through millions of samples every iteration will be too costly in terms of time and computing power.

- *Effect of normalization/standardization on the data*

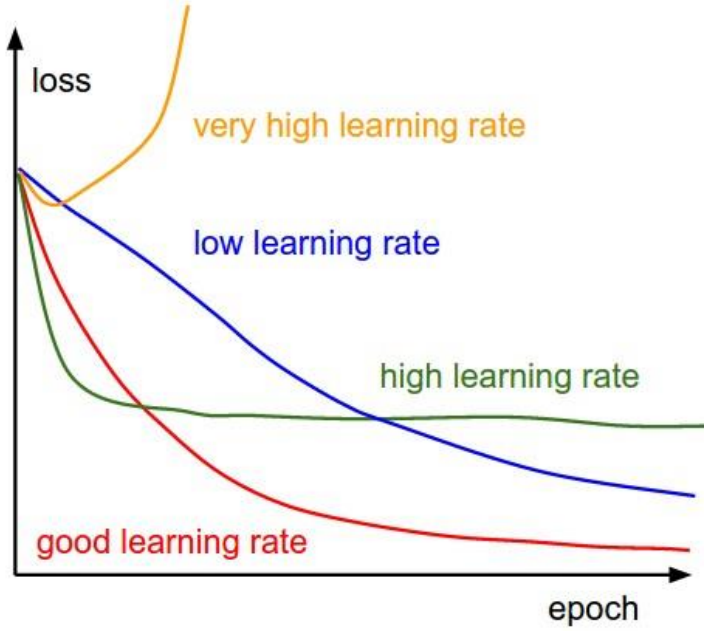
  Copying our gradient descent equation from above:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

  The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.
  Having features on a similar scale will help the gradient descent converge more quickly towards the minima.

- *Effect of training iterations on loss*

  Effect of training iterations on loss can vary quite a lot and depends on other factors such as learning rate as well. Generally, with a good learning rate we should see an inverse relation between training iterations and loss i.e. increasing iterations should result in decreasing loss.



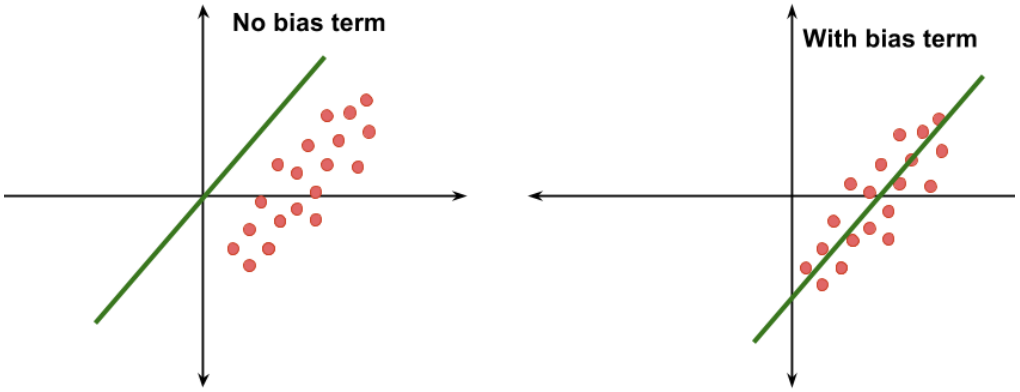
  As we can see choosing a good learning rate will help in reducing loss with increasing iterations but a bad choice of the learning rate can even result in divergence of loss.

  Also, even with a good learning rate we can see in the graph that after a certain number of iterations the loss function flattens out i.e. the model has reached the minima and there is not much further scope of improvement. So, keeping large value of iterations (say, ~10^10) is also not fruitful as it takes a lot of time and we stop reducing loss after a certain number of iterations anyway.

- *Difference between GD and SGD plots*

  According to the plots obtained we observe that convergence is smooth for GD at lower values of learning rate(alpha ~ 0.0001) whereas SGD performs better when we take a higher learning rate(alpha ~ 0.01) coupled with higher number of iterations(more passes through the data).

- *Would excluding the bias term from our model reduce the error*
  Removing the bias term and running the gradient descent algorithm gives us a mean error of around 0.868 which is a bit better than with the bias term. But in general we should definitely always include the bias term otherwise if our features are 0, target also has to be 0 which might not always be the case. For our data features never take all 0 values hence it didn't matter.
  One scenario in which ignoring the bias term might cause problems is depicted below:



- *Feature importance*
  Weight vector gives us the significance/importance of each feature variable in our data.
  Observed values:
  Theta = [1.37e-17 0.2784 0.1672 0.0541]
  As we can see from the coefficients feature importance is in this order:
  bmi > age > #children