CMPT726 - Machine Learning
Assignment 3

Topic: Logistic Regression and Neural Networks
Submitted to: Professor Oliver Schulte
Submitted by: Raquel Yuri da Silveira Aoki
Date: November 17, 2017

Question 1 - Gradient Descent Update for Logistic Regression Using Cross-Entropy

For a probabilistic classifier, it is natural and effective to take the negative class label log-likelihood as the empirical loss function, which is known as the cross-entropy loss. The cross-entropy is defined as follows. Let $p_w(y = 1|x)$ be the probability that the class label $y = 1$ given input features x, where $y = 0, 1$. The cross-entropy of data (x_j, y_j) is then defined as

$$Loss(w) = -\frac{1}{N}\sum_{j=1}^{N} y_j \times ln(p_w(y_j = 1|x_j)) + (1 - y_j) \times ln(1 - p_w(y_j = 1|x_j))$$

Notice that either y_j or $1 - y_j$ is 0. For logistic regression, the conditional probability is $p_w(y_j|1 = x) = \frac{1}{1+e^{-wx}}$

1) Write down the cross-entropy loss for logistic regression.
Solution

$$Loss(w) = -\frac{1}{N}\sum_{j=1}^{N} y_j \times ln(\frac{1}{1 + e^{-wx}}) + (1 - y_j) \times ln(1 - \frac{1}{1 + e^{-wx}})$$

$$Loss(w) = -\frac{1}{N}\sum_{j=1}^{N} y_j \times ln(\frac{1}{1+e^{-wx}}) + (1 - y_j) \times ln(\frac{e^{-wx}}{1+e^{-wx}}) \text{ (Eq. 1)}$$

2)Prove that for logistic regression, the cross-entropy gradient is

$$\nabla Loss(w) = \frac{1}{N}\sum_{j=1}^{N}(p_w(y_j = 1|x_n) - y_j)x_n$$

Solution

$Loss(w) = -\frac{1}{N}(Y \times ln(\frac{1}{1+e^{-WX}}) + (1 - Y) \times ln(1 - \frac{1}{1+e^{-WX}}))$
$Loss(w) = -\frac{1}{N}(Y \times ln(\sigma(wx)) + (1 - Y) \times ln(1 - \sigma(wx)))$

Then:

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Y\frac{1}{\sigma(wx)}\sigma'(wx)x_n + (1-Y)\frac{1}{1-\sigma(wx)}(-1)\sigma'(wx)x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Y\frac{1}{\sigma(wx)}\sigma(wx)(1-\sigma(wx))x_n + (Y-1)\frac{1}{1-\sigma(wx)}\sigma(wx)(1-\sigma(wx))x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Y\frac{1}{\sigma(wx)}\sigma(wx)(1-\sigma(wx))x_n + (Y-1)\frac{1}{1-\sigma(wx)}\sigma(wx)(1-\sigma(wx))x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Y(1-\sigma(wx))x_n + (Y-1)\sigma(wx)x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Yx_n - Y\sigma(wx))x_n + Y\sigma(wx)x_n - \sigma(wx)x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = -\frac{1}{N}(Yx_n - \sigma(wx)x_n)$

$\frac{\partial Loss(w)}{\partial w_n} = \frac{1}{N}(\sigma(wx) - Y)x_n$

$\frac{\partial Loss(w)}{\partial w_n} = \frac{1}{N}\sum_{j=1}^{N}(\sigma(wx) - y_j)x_n$

$\frac{\partial Loss(w)}{\partial w_n} = \frac{1}{N}\sum_{j=1}^{N}(p_w(y_j = 1|x_n) - y_j)x_n$

3) Following the notation in the text, write down the stochastic gradient descent update formula for minimizing the cross-entropy with logistic regression.

Solution

$$w_n \leftarrow w_n - \alpha(p_w(y_j = 1|x_n) - y_j)x_n,$$

where α is the learning rate.

Question 2 - Implementing Logistic Regression

Use the data set as before, with the modifications indicated on-line. Using $GP > 0$ as the target class variable and drop the sum_7yr_GP column. We will go through a few typical steps for a regression analysis: data preprocessing, weight learning, and model evaluation.

Data Preprocessing; This is very similar to what we did in Assignment 2. For completeness and clarity, I repeat some instructions.

Convert discrete to continuous: The hockey data form a hybrid data set, meaning that it mixes discrete with continuous variables. Regression is a technique for continuous input variables. A common approach to using regression with discrete variables is to first convert all discrete variables to binary (Boolean) variables, then use 1 and 0 to represent variables. Boolean variables that are treated as continuous regressors are called " variables or indicator variables. So the first step is to replace all discrete variables by dummy variables. For example, instead of having a single variable Country Group with three possible values Canada, USA, Europe, you should introduce three binary variables Country Group = Canada, Country Group =

USA, Country Group = Europe. Change the data matrix so all values for the dummy varrables are 1 or 0, depending on where the player is from.

Standardize Input Variables: As discussed in class, in a multivariate regression it is often a good idea to standardize continuous variables to the same scale. In this assignment, we use gradient descent with a single step size, so the input variables should be on a single scale.

Implement Stochastic Gradient Descent To reduce your workload, the code we have posted on the website does almost everything you need to implement. If you prefer, you can also build your own implementation and use the posted code for guidance only. Given that you can build on your solution from Assignment 2, this should not be too much work. Here are some specific components we will check.

1. In the posted code, fill in the part that performs a single gradient descent update, using your solution formula from the previous question. If you submit your own code, please highlight the part that performs the single gradient descent update.

2. Adjust the provided code (or build your own) to perform stochastic gradient descent with the hockey dataset. (E.g., you need to fix some of the dimensions that are hardcoded into the code. And you may want to change some of the notation). For the initial weight vector, set the bias weight $w_0 = 0, 1$ and all others to 0.

3. Write code that takes as input a weight vector w and a dataset and outputs the following:

(a) The logistic regression cross-entropy (negative log-likelihood). (This is already implemented in the sample code.)

(b) The classification accuracy of w: Label an example x as positive if the predicted probability $p_w(y = 1|x) > 0.5$, and as negative otherwise. Then output the percentage of correctly labelled examples.

Experiments: Congratulations, you have a working implementation that learns weights for logistic regression. Now we can try things out. Try three different step sizes (aka learning rates), $\alpha = 0.5, 0.3, 0.1, 0.05, 0.01$, each for 300 training epochs, i.e. 300 passes through the entire dataset (batches).

1 For each learning rate, plot a learning curve that shows the cross-entropy after each epoch. So the x-axis is the epoch number and the y-axis is the cross-entropy (negative log-likelihood) for that epoch. Put the learning curve for all rates into a single plot and compare the results. What are the advantages and disadvantages of different learning rates? Figure 18.25 in the text

shows you an example of a similar plot for a single learning rate

Solution

Figure 1 shows the negative log likelihood of SGD using cross-entropy on Logistic Regression. In this dataset the SGD converge very fast and needs at most 15 epochs. The lower values were with learning rates of 0.01 and 0.05. In this problem, an advantage of lower values is that they converge quicker than large values (such as 0.5). On the other hand, using lower values increase the chance of stop on a local minimum.

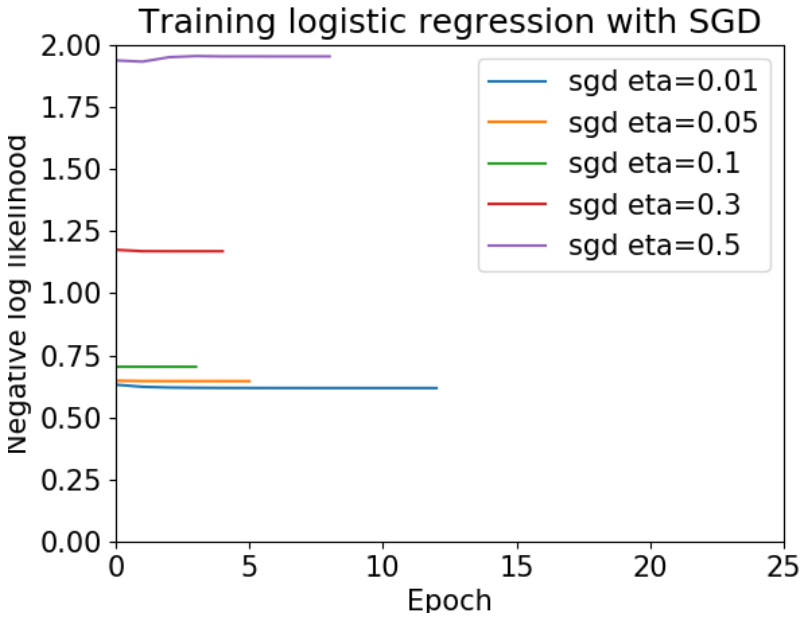

Figure 1: Negative log-likelihood

2 From your plot, find the learning rate that converges to the weight vector w with the best fit to the training data (lowest cross-entropy). What is the accuracy of w on the test data? Using your results from previous assignments, compare the classification accuracies of Naive Bayes, decision tree ID3 and logistic regression.

Solution

Figure 2 shows the accuracy versus learning rates of this problem. The accuracy of those learning rates is almost the same, about 61%. In assignment 1, the accuracy of Naive Bayes was 59.68% and in assigment 2, the accuracy of the Decision Tree was 70.15%. So, the result of this model is similar to Naive Bayes, and Decision Tree still a better predictor to Hockey Draft Dataset.

Extra

Solution

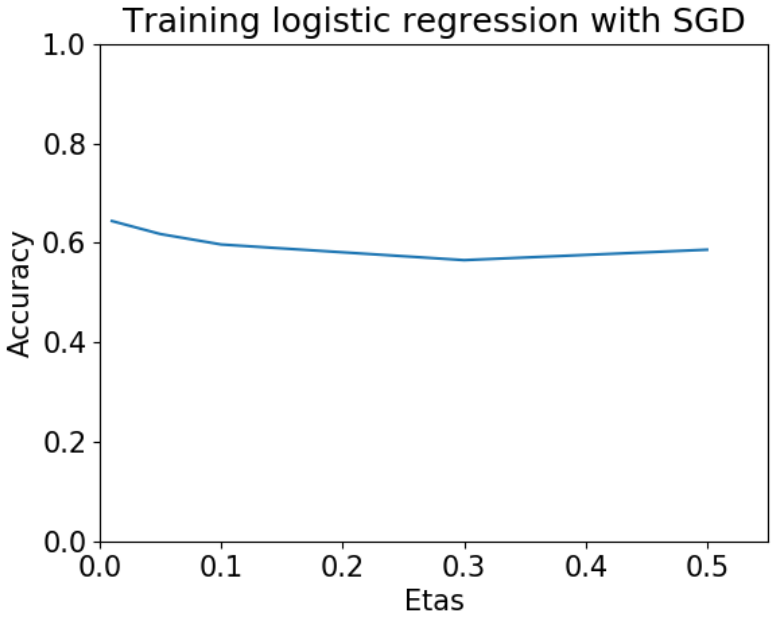

Figure 2: Accuracy versus learning rates

Use gradient descent does not improve the results and both are very fast. The gradient descent implementation can be checked on the code. The use of original features rather than the standardized ones decrease the accuracy. It was not test add new iterations, but it was test remove some features and this also doesn't increase the accuracy.

Question 3 - Trace Backpropagation

Consider a neural net with one hidden layer, two inputs a and b, one hidden unit c, and one output unit d. The activation function is the sigmoid function for each node. This network has five weights $(w_{ac}, w_{bc}, w_{0c}, w_{cd}, w_{0d})$, where w_{0x} represents the bias or threshold weight for unit x. Initialize these weights to the values $(.1, .1, .1, .1, .1)$, then give their values after each of the first two training epochs of Backpropagation algorithm. Assuming learning rate (step size) of 0.3, stochastic gradient descent, and the following training examples:

Table 1: Data

	a	b	d
x_1	1	0	1
x_2	0	1	1

Fill in the following tables. The tables use the notation from the slides. You can expand these to include more information (derivates of activation functions) if you like.

Solution
Neural network design:

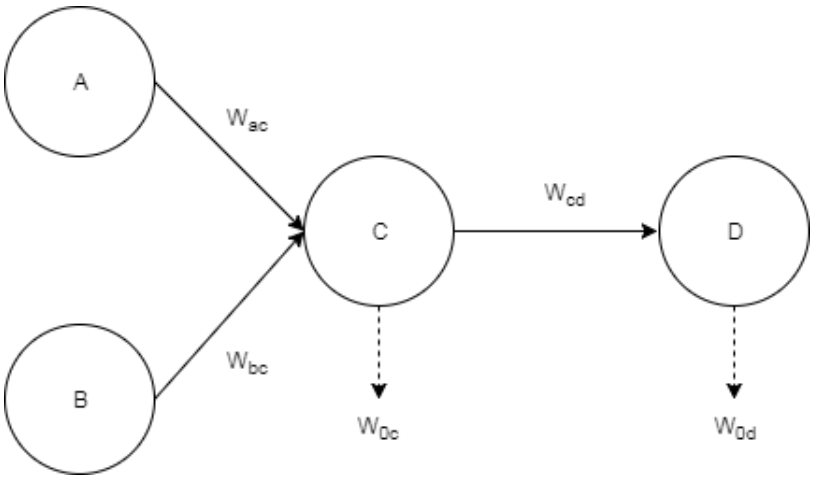

Figure 3: Neural Network

The elements a_c, a_d, δ_c, δ_d are defined as:

- $a_c = g(in_j) = \sigma(in_j) = \sigma(w_{ac} \times x_a + w_{bc} \times x_b + w_{0c})$
- $a_d = g(in_k) = \sigma(in_k) = \sigma(w_{cd} \times a_c + w_{0d})$
- $\delta_d = g'(in_k)(y_k - a_k) = \sigma(w_{cd} * a_c + w_{0d}) \times (1 - \sigma(w_{cd} * a_c + w_{0d})) \times (x_d - a_d) = a_d \times (1 - a_d) \times (x_d - a_d)$
- $\delta_c = g'(in_j) \sum w_{jk}\delta_k = a_c \times (1 - a_c) \times (w_{cd}\delta_d)$

where $\sigma(z) = \frac{1}{1+e^{-z}}$, $\sigma'(z) = \sigma(z) \times (1 - \sigma(z))$, $in_j = w_{ac} \times x_a + w_{bc} \times x_b + w_{0c}$ and $in_k = w_{cd} \times a_c + w_{0d}$. In order to update the weights, it is used the follow results:

- $\frac{\partial E_{total}}{\partial w_{0d}} = \left(\frac{\partial 0.5(x_d - a_d)^2}{\partial a_d}\right) \times \sigma'(in_k) \times \left(\frac{\partial (in_k)}{\partial w_{0d}}\right) = (x_d - a_d) \times \sigma(in_k)(1 - \sigma(in_k)) \times 1 = \delta_d$
- $\frac{\partial E_{total}}{\partial w_{cd}} = \delta_d * a_c$ (same logic used with $w_0 d$)
- $\frac{\partial E_{total}}{\partial w_{0c}} = \left(\frac{\partial E_t}{\partial a_c}\right) \times \sigma'(in_j) \times \left(\frac{\partial in_j}{\partial w_{0c}}\right) = \left(\frac{\partial E_t}{\partial in_k}\frac{\partial g(in_k)}{a_c}\right) \times \sigma'(in_j) \times 1 = \left(\left(\frac{\partial E_t}{\partial a_d}\frac{\partial g(in_k)}{\partial in_k}\right)\frac{\partial g(in_k)}{a_c}\right) \times \sigma'(in_j) \times 1 = ((x_d - a_d)\sigma'(in_k)w_{0d}) \times \sigma'(in_j) \times 1 = \delta_d w_{0d} \times \sigma'(in_j) \times 1 = \delta_c \times 1$
- $\frac{\partial E_{total}}{\partial w_{ac}} = \delta_c \times x_a$ (same logic used with $w_0 c$)
- $\frac{\partial E_{total}}{\partial w_{ac}} = \delta_c \times x_b$ (same logic used with $w_0 c$)

Thus:

- $w_{0d} \leftarrow w_{0d} + \alpha \times \delta_d$
- $w_{cd} \leftarrow w_{cd} + \alpha \times \delta_d \times a_c$
- $w_{0c} \leftarrow w_{0c} + \alpha \times \delta_c$
- $w_{ac} \leftarrow w_{ac} + \alpha \times \delta_c \times x_a$
- $w_{bc} \leftarrow w_{bc} + \alpha \times \delta_c \times x_b$

backpropagation trace:

	a_c	δ_c	a_d	δ_d
x_1	0.5498	0.0028	0.5386	0.1146
x_2	0.5500	0.0032	0.5497	0.1114

	w_{0c}	w_{ac}	w_{bc}	w_{0d}	w_{cd}
x_1	0.1008	0.1008	0.1	0.1343	0.1189
x_2	0.1018	0.1008	0.1009	0.1678	0.1862

Question 4 - Vanishing Gradients

Consider a simple linear neural net with the structure *input* $\rightarrow$ *node*1 $\rightarrow$ *node*2 $\rightarrow$ *output*, as shown in Figure 1. There are 6 weight parameters; assume they have all been initialized to 0.1. Apply backpropagation to the training data (1, 1), input =1 is mapped to output = 1.

- Assume a sigmoid activation function for each node (other than the input). Write down the gradient for each weight parameter for the training data (1, 1)
- Assume a rectified linear unit activation function for each node (other than the input). Write down the gradient for each weight parameter for the training data (1, 1).

Which activation function leads to higher gradients? What if we extended the network with additional layers (intermediate nodes in the line). Would the difference in gradients between sigmoid and ReLU activations become greater or smaller?

Solution

Figure 4 is a illustration of the neural network and below there is a summary which equations, weight updates, gradient values. The Rectified linear unit (ReLU) leads with higher gradients then sigmoid activation function. If the network be extended, this difference will increase because each δ_{Ni} use the δ_{Ni-1} from the previous layer. So, even though this values are very small, after the product they will contribute for a bigger difference between the gradients from sigmoid and ReLu activations.

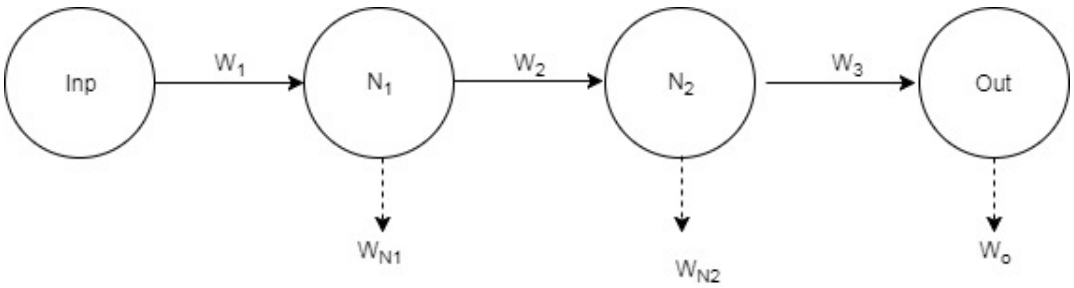

Figure 4: Neural Network

	Start		Sigmoid	ReLU
N_1	$g(w_1*x+w_{N1})$		0.5498340	0.7981389
N_2	$g(w_2*N_1+w_{N2})$		0.5386685	0.7870903
Output	$g(w_3N_2+w_O)$		0.5383910	0.7864885
δ_O	$g'(w_3N_2+w_O)(y\text{-}Ouput)$		0.1147219	0.1162695
δ_{N2}	$g'(w_2N_1+w_{N2})*w_3*\delta_O$		0.0028509	0.0063347
δ_{N1}	$g'(w_1*x+w_{N1})*w_2*\delta_{N2}$		0.0000706	0.0003483
w_1	$w_1+\alpha*\delta_{N1}*x$	0.10	0.1000071	0.1000348
w_2	$w_2+\alpha*\delta_{N2}*N_1$	0.10	0.1001568	0.1005056
w_3	$w_3+\alpha*\delta_O*N_2$	0.10	0.1061797	0.1091515
w_{N1}	$w_{N1}+\alpha*\delta_{N1}$	0.10	0.1000071	0.1000348
w_{N2}	$w_{N2}+\alpha*\delta_{N2}$	0.10	0.1002851	0.1006335
w_O	$w_O+\alpha*\delta_O$	0.10	0.1114722	0.1116270
x		1.00	g(z)=1/(1+exp(-z))	g(z)=ln(1+exp(z))
y		1.00	g'(z)=g(z)*(1-g(z))	g'(z)=1/(1+exp(-z))

Question 5 - Deep Learning

Install a deep learning package. I have listed a number on the course website. Most students have liked Keras and Caffee (which interfaces with Torch). Try to use your package to predict the number of games an NHL player will have played after 7 years. (The same problem we tackled before with linear regression). For data preprocessing, drop the $GP > 0$ column and use sum_7yr_GP as the target column. Try to find deep learning settings for training on the test set such that the trained model gives the best prediction

on the test set. (In terms of the textbook discussion, we are using the test set as a validation set for hyperparameters.) There will be a prize for the student(s) who get the lowest squared error on the test set! You should use a linear activation function for the output node. (Why not sigmoid?) There are some settings you must try and some that I suggest you try.
Solution

- To solve this problem sigmoid linear activation function is not a good option because the output is not between 0 and 1. Thus, an activation function that does not limit the output, such as linear, is a better option.

- The packages used in this question was $keras+tensorflow$. The parameters were $nb\_epoch = 200$, $validation\_split = 0.15$ and $verbose = 50$

- 1296 combinations of neural networks were tested, with 0, 1 or 2 hidden layers, and 6, 12, 24 or 48 units each layer, and ReLU, linear or sigmoid activation functions. From those combinations it is possible extrac the follow results:

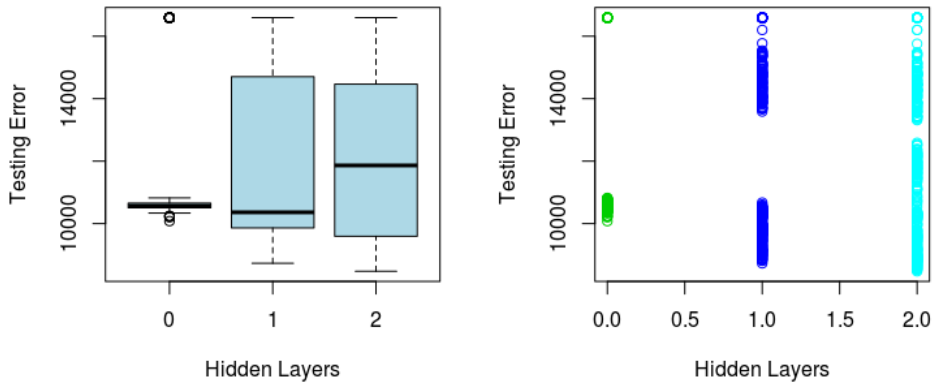

Figure 5: Comparison between number of hidden layers - Testing Error

  In average, neural networks with 0 hidden layers have smaller error. However, the minimum error occur with neural networks with 2 hidden layers, as shown in Figure 5.

- Considering the top 5 models shown on Table 2, the ideal number of units in each hidden layers varies between 6 and 12.

DN	Testing Error	Hidden Layers	Units	Activation Funct
1	8472.360	2	12,6	'linear', 'relu', 'relu'
2	8511.505	2	6, 6	'relu', 'linear', 'relu'
3	8531.972	2	12, 12	'linear', 'linear', 'relu'
4	8562.112	2	12, 6	'linear', 'linear', 'relu'
5	8569.156	2	6, 6	'relu', 'relu', 'linear'

Table 2: Top Deep Models

- The best results have ReLU as activation function on the external layer. On first hidden layer the best combination is $12 units + linear$ or $6 units + ReLU$; on the second hidden layer, the best results are with $6 units + ReLU or linear$. Deep networks that use in some hidden layers sigmoid doesn't present good results.
- One technique to avoid overfitting is dropout. It was tested a dropout with 0.1 and 0.2, and using standard variables, the droupout(0.1) presented better results than models without dropout.

Figure 6 compare the test set with the predictions made by deep learning model and linear regression model (assignment 2). It is possible to notice that both models have a similar performance oon the test set.

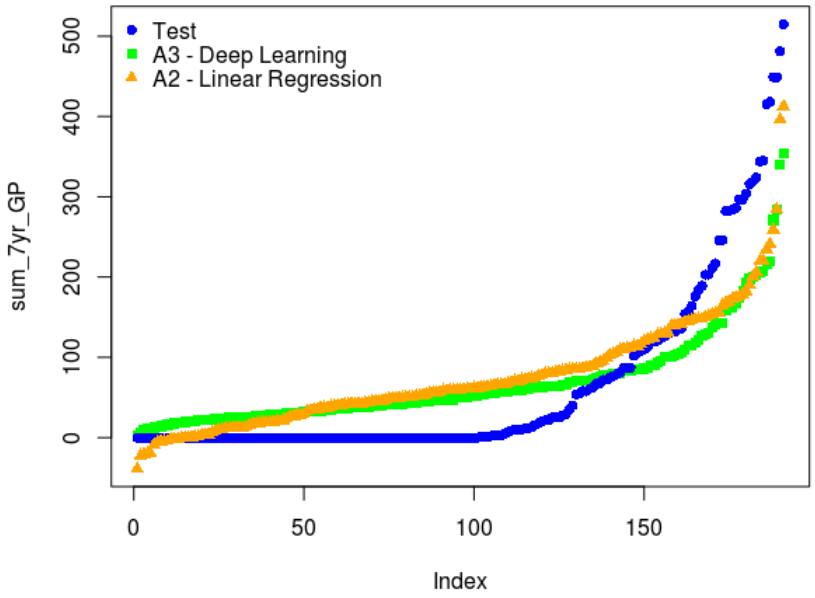

Figure 6: Comparison between methods