MACHINE LEARNING

UNIVERSITÀ DELLA SVIZZERA ITALIANA

Assignment 1

LIBERATI JACOPO

March 26, 2024

Task 1

$f(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{0.96551392} + \mathbf{5.05153814}\cdot x_1 \mathbf{-4.00489622}\cdot x_2 + \mathbf{7.02109756}\cdot\cos(x_1) + \mathbf{1.99768997}\cdot x_2\cdot x_2 - \mathbf{0.10373533}\cdot\tanh(x_1)$

In the first task, after importing the necessary libraries and importing the data, the dataset (data) was recreated based on the family of models provided.
Subsequently, a split of the data was carried out, keeping 70% of the data for the train and 30% of the data for the test.
Subsequently, the train set was used to carry out the linear regression and estimate the required parameters. The parameters were found by solving the closed formula that we can use for Linear Models.
Subsequently, the code for parameter evaluation was reported using the LinearRegression module from the sklearn.linear_model library, which can be used to identify the same results found with the first proposed method.
Subsequently, both the performance on the train set and the performance on the test set were calculated using the MSE. As expected, the MSE on the train set is lower than that on the test set, although the difference is not large, demonstrating the quality of the model.
Subsequently, the Lasso Regression was carried out, importing the relevant module from the sk.learn library. Using the same data split used in the previous point, the model coefficients and also the performance on the train and on the test set were calculated again, obtaining in this case too, as expected, an MSE on the train set lower than that on the test set.
In addition to this, it is observed that through the Lasso regularization there is a reduction in the complexity of the model, since the last coefficient is brought to 0.
From this it can be deduced that the variable associated with the last parameter has a negligible impact on the prediction of the model. In fact, through Lasso, we have the addition of a penalty which leads the unimportant coefficients to be exactly 0.
Despite this, the values of the train error and test error are similar to those found in the case of linear regression, thus showing the quality of the model and family of models previously used. However, a reduction in the test error is noted, since through Lasso it is possible to eliminate the overfitting which could however be partially present within a linear model, due to redundant data within the dataset.

Task 2

In the second task a non-linear family of models was chosen and the performance of the model was subsequently calculated again on the basis of the MSE. A Random Forest was selected, a model that combines decision trees, each of which can capture any non-linear relationships in the data. After importing the RandomForestRegressor module from the sklearn library, using the same data split carried out at the beginning, test performance and train performance were calculated again through the MSE. Comparing this result with those obtained for the initial linear model, in this case we find a much lower train error than the previous model, while the test error is higher than the previous model.

This indicates that the Random Forest is overfitting on the training data; the model is likely too complex and over-fits the training data, also capturing noise in the data rather than underlying patterns.

This leads to high precision on training data, but poor generalization on new data, confirmed by a higher test error than the previous one. Furthermore, the random forest proves to be more capable of storing training data rather than learning generic patterns in the data.

Subsequently, a t-test on the residuals was carried out to statistically compare the performance of the linear regression and the random forest. For both models, the residuals were calculated as the difference between the observed values and the predicted values. Subsequently the stats module was imported from the scipy library, in order to calculate the T-statistic and the p-value.

The p-value is higher than the established significance level (0.05) and this implies that there is no significant difference between the two models. This is confirmed by a T-statistic value that is in the range between -1.96 and 1.96, which are the critical values considering a Gaussian distribution with a 95% confidence interval. In this case, we do not have sufficient evidence to reject the null hypothesis and therefore we conclude that there is no statistically significant difference between the two models.

We can conclude this, despite the linear model being less complex and having a lower MSE on the test set than the random forest.

Task 3 (Bonus)

In task 3 it is required to identify a model that has a lower MSE than that of the proposed baseline model (0.013). In response to this, code was provided to train and evaluate a regression model using a sequential neural network with TensorFlow. This allowed me to obtain a lower MSE on the test set, 0.0124. The model was subsequently saved as 'sequential_model_t3.pkl'.

After importing data_bonus.npz, this was split into train and test set.

Subsequently a sequential model was created with a series of layers, each of which with a specific activation function. The activation functions used are non-linear activation functions, such that they can allow the model to learn the non-linear relationships between the data.

The last output layer is characterized by a single neuron and the activation function is linear, since it is a regression problem. The optimizer used (Adam) and the loss function used to evaluate the model performance (MSE) are then specified.

Subsequently the model is trained (choosing epochs, batches and data to use as validation set) and finally evaluated on the basis of the test set.

The best performance of the model can be attributed to the high number of neurons used for each layer, the non-linear activation functions and the dropout technique included within the model.

QUESTIONS

Q1. Training versus Validation

Q1.1 What is the whole figure about?

A1.1 In this figure we have a representation of how the performance varies as the complexity of the model varies. In particular, the complexity of the model can be represented by the number of parameters that must be estimated.

As a first fundamental element, we can distinguish between Test Error and Training Error:

→ Considering the Training Error, we see that as the complexity of the model increases, the training error goes to 0, meaning that we have sufficient degrees of freedom to learn exactly every single point that is inside the training set.

However, Vn(ϑ) is not the Performance at Task.

→ An unbiased estimator of the Performance at Task is the Test Error and we see that if we use a model that is too simple, the performance will be bad, but also if we use a model that is too complex, the performance on the test set will be bad, because we are able to go exactly through the points inside the training set, but this doesn't imply that we'll be good at task. If we receive points in the Test Set that are different from those used for training, the performance will be bad.

This is the reason why it is necessary to identify an excellent complexity of the model, which allows us to minimize the Test Error, unbiased estimator for the Performance at Task.

Q1.2 Explain the behaviours of the curves in each of the three highlighted sections inthe figure, namely (a), (b), and (c).

A1.2 After explanation provided in the previous answer, we can add that when we see the line of the Training Error going up and down, it means that this is the training error that we are going to observe, this is not an expectation.

When we have the smooth curve, this is the expectation over different training data of the same size.

Also, when we look at the Validation Error, we see again something going up and down.

As we'll see later, what we have to do is to take the minimum ϑm and we consider this to be the optimal configuration of parameters, even if we don't bring the gradient to 0.

The Expected Test Error is the smooth line and we know that when we have ϑm, on the red curve we'll have a point, which is the corresponding Test Error.

Q1.2.a Can you identify any signs of overfitting or underfitting in the plot? If yes,explain which sections correspond to which concept.

A1.2.a Talking about the concept of **UNDERFITTING** and **OVERFITTING**:

→ if the model is less complex that the optimal one, we are underfitting the data, meaning that we don't have enough degrees of freedom to explain the information content coming from the data; we are to the left of the optimal model complexity (figure a).

→ if the model is more complex that the optimal one, we are overfitting the data, meaning that we are fitting too much our training points, the experience; we are to the right of the optimal model complexity (figure c).

Q1.2.b How can you determine the optimal complexity of the model based on thegiven plot?

A1.2.b To understand the optimal model complexity, we can use **EARLY STOPPING**, a method allowing us to control the parameters and try to find something whose complexity is optimal.

We want to learn a function g(x), but we don't know its complexity.

If this is complex, we'll need a lot of neurons, a huge model complexity; if this is a simple task, we'll need a linear model, few parameters.
If we don't know how to solve this problem, we can consider a huge model with many parameters, so that it will have enough expressive power to solve g(x).
In particular, we need to iterate the Training Procedure and we opt for a Back propagation, iterating it over time:

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}\bigg|_{\theta=\theta_i}$$

The iteration is a way to control the complexity of the model, by moving over the curves.
Even if we have many degrees of freedom, the true complexity of the model at the beginning is small because we are not using the degrees of freedom in a nice way to solve the problem.
The complexity is not only the number of parameters, but it's also the effective number of parameters that the model is using to solve the task.
By learning over time, we'll explore the plot, from the left to the right and what we want to do is to stop where we have the optimal test error, which is unbiased estimator of the performance at task.
However, we can't use the test set to drive ourselves in when to stop, because if we proceed in this way, we can't use the test error in assessing the ability to solve the task.
If we decide when to stop on the basis of the data on which we will be tested, then the test set is not a correct measure of understanding anymore.
This is why we have to introduce the Validation Set, which is the set of data on which we decide when to stop the reiteration.
After that, the test will be done on a new set of fresh data, getting the Test Error, which is now unbiased and can be used to estimate the performance at task.

$$V_l(\hat{\theta}) = \frac{1}{l}\sum_{i=1}^{l} (\tilde{y}_i - \tilde{x}^T \hat{\theta})^2$$

Q1.3 Is there any evidence of high approximation risk? Why? If yes, in which of thebelow subfigures?
A1.3 The approximation risk represents the performance of the model on unseen data, how good the model is at approximating data that do not belong to the training set.
In particular, this is linked to the complexity of the model because when the model is too little complex, we have a bad performance related to data belonging to the training set and even worse against new data not belonging to the training set.
However, when the complexity of the model is too high, despite having a training error tending to 0, the model still has a weakness in approximating new data different from that used for training.
Consequently, the risk approximation is evident in figure a and figure c, which share a test error and a validation error that remain high.
The model is better able to approximate fresh data only when we select an optimal model complexity.

Q1.4 Do you think that increasing the model complexity can bring the training error to zero? And the structural risk?
A1.4 The training error is very biased on the data that we have and it is strongly related to the model complexity. If we use a very large model, the training error will go to 0, because we are able to learn both g(x) and the noise, meaning that we are able to pass through the points that we have in our training set.
On the other hand, the structural risk represents the performance at task, that is not to be able to pass through the points we have in our training set, but to be able to approximate also fresh data.

We want to learn g(x) and not also the noise that we have in the process.

$$\overline{V}(\theta) = \int L(y, f(\theta, x)) p_{x,y} dxy$$

The structural risk is assessing our model over the entire space, by assuming that we have infinite points affected by uncertainty. However, we can't minimize the structural risk because we don't know the joint probability density function between x and y and also, we can't compute the integral because we have only a finite set of n points.
Because of this, we try to estimate the performance at task (the structural risk) through the test set, that is an unbiased estimator of the structural risk.
Coming back to the question, given that as we increase the model complexity the test error increases, also the structural risk will increase.
Increasing the model complexity, we'll be able to approximate better and better the point that we have inside our training set, but we'll be less able to approximate in a good way fresh data not used for training.

Q1.5 If the X axis represented the training iterations instead, would you think that the training procedure that generated the figure used early stopping? Explain why. (NB: ignore the subfigures and the dashed vertical lines)

A1.5 If on the X axis we visualize the number of reiterations, we can use early stopping, which as mentioned is a method through which we can identify the optimal model complexity.
Considering a huge model, with many parameters, we'll have enough expressive power to solve g(x). However, looking at the relation between performance and model complexity, with a huge model, the performance will be very small (overfitting) and we'll be very bad in the test set.
However, the point of early stopping is that through the training procedure (back propagation), that is reiterated over time, we explore this plot and we have the possibility to move toward the right, with the objective of stopping around a point where we have the optimal test error.

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}\bigg|_{\theta=\theta_i}$$

Through this mechanism, we'll decide when to stop, on the basis of the Validation Set, that is what is done looking at the figure, where we stop in a point that is before the minimum of the Expected Test Error, meaning that the gradient doesn't go to 0.

Q2. Linear Regression

Comment and compare how the (a.) training error, (b.) test error and (c.) coefficientswould change in the following cases:

Q2.1 $x_3 = x_1 + 0.2 \cdot x_2$.

A2.1

a) Training Error: we are adding a new regressor that is a combination of X1 and X2.
This regressor might improve the model's ability to fit the training data since it introduces a new feature that captures additional information. Because of this, the training error might decrease.
b) Test Error: the effect on the test error depends on how well this new feature captures the underlying relationship between X and Y. If X3 provides valuable information not captured by X1 and X2, it may reduce the test error. However, if X3 introduces noise or redundancy, it could increase the test error.
c) Coefficients: depending on the correlation between X1, X2 and X3, θ3 might take on a significant value, reflecting the importance of the new feature in predicting Y.

Q2.2 $x_3 = x_1$^2 (in Python ^ is the "power" operator, so 3^ 2 = 3 * 3 = 9).A2.2

a) Training Error: we are adding a new regressor that introduces non-linearity to the model, potentially improving its fit to the training data. Hence, the training error might decrease.

b) Test Error: we can say if the test error decreases or increases depending on the true relationship between X and Y.
If the true relationship is nonlinear and closer to quadratic, adding this quadratic term may improve the model's ability to generalize, reducing the test error. However, if the true relationship is linear or follows a different non-linear pattern, adding this term may increase the test error.
c) Coefficients: The coefficient of X3 ($\theta 3$) captures the influence of the quadratic term of X1 on the target variable. If the relationship between X1 and Y is quadratic, $\theta 3$ will be non-zero and reflect the importance of this quadratic term in predicting Y.

Q2.3 x_3 is a random variable independent from y.
A2.3
a) Training Error: the addition of a random variable that is independent of the target variable probably will introduce noise into the model, potentially increasing the training error.
b) Test Error: since X3 is unrelated to Y, adding it as a feature would not provide any meaningful information for predicting Y, and this will increase the test error.
c) Coefficients: the coefficient of X3 ($\theta 3$) would ideally be close to zero since X3 is independent of Y. Due to the noise, it's still possible that $\theta 3$ is not exactly zero, but it will be compared to coefficients associated with meaningful features.

Q2.3 How would your answers change if you were using Lasso Regression?
A2.3
First Case: Lasso regression tends to select only a subset of features by driving some coefficients to exactly zero. If the combination X1+0.2*X2 is not essential for predicting the target variable, Lasso regression might drive the coefficient of X3 to zero, disregarding this feature.
However, if this combination captures important patterns, Lasso may retain it in the model.
Second Case: Lasso regression might set the coefficient of X3 to zero if the quadratic term does not significantly improve predictive performance.
However, if the quadratic term captures crucial nonlinear relationships, Lasso might retain it, especially if it reduces the overall loss sufficiently to compensate for the regularization penalty.
Third Case: Lasso regression tends to drive irrelevant coefficients towards zero. If X3 is entirely unrelated to Y, Lasso will probably set its coefficient to zero, effectively ignoring this feature.

Q2.4 Explain the motivation behind Ridge and Lasso regression and their principal differences.
A2.4 Behind Ridge Regression and Lasso Regression there is the idea that we want somehow to minimize the number of measurements.
If something is not relevant to our problem, we want this parameter to go to 0 and not contribute to the solution of the problem, because it means only to increasing our costs.
RIDGE-REGRESSION:
Ridge regression aims at pushing as many parameters as possible towards zero by adding a shrinking penalty to the loss function. Inputs are centered.
If this was our MSE Training Function:

$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - x_i^T\theta\right)^2$$

We modify Vn(θ) in VRidge(θ):

$$V_{Ridge}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - x_i^T\theta\right)^2 + \lambda\sum_{i=2}^{d}\theta_i^2$$

In the Ridge Regression we have 2 terms:

→ The first one is associated with the performance of our model.
→ We introduce a penalty term with λ that is a positive number because we want to penalize those weights that are not necessary.
It means that if we can put towards 0 something, we will improve over the loss function.
The function that we have to minimize is the following:

$$V_{Ridge}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - x_i^T\theta\right)^2 + \lambda\|\theta\|^2$$

And if we do that, the solution under the linear assumption of our problem is the following:

$$\hat{\theta}_1 = \frac{1}{n}\sum_{i=1}^{n} y_i \qquad \hat{\theta} = \left(X^TX + \lambda I\right)^{-1}X^TY$$

If we look at the Ridge Regression there is a TRADE OFF, as we can see there is a "+":
The tradeoff is between the performance (Training Procedure) and the fact that we want to minimize this squared magnitude of the parameters.
What we expect through the Ridge Regression is that all those parameters which are not relevant to the problem, will naturally go towards 0, we can get rid of them.

LASSO:
When we take this into account, a method which has the same idea of pushing to 0 the parameters which are not relevant is called LASSO.
In Lasso we substitute the Square of the Parameters with the Magnitude of the Parameters because we want to minimize the parameter, what has to go to 0 is the parameter and not its square.
This is due to the fact that if we have a parameter that is less than 1 and we square it, it will go toward 0 more quickly and faster, but we don't want to do that.
We change the second term and we put the absolute value of the parameter.

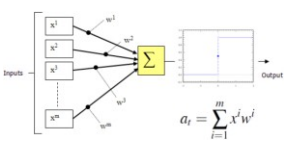
$$V_{Lasso}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - x_i^T\theta\right)^2 + \lambda\sum_{i=2}^{d}|\theta_i|$$

In the Ridge Regression, it will converge toward something that is more or less 0.
Here it will converge to 0 or something which is not 0; if it goes to 0 we can get rid of the parameter.
However, in order to solve this problem, WE CAN'T DO IT IN CLOSE FORM, while with the Ridge Regression we can do it.
If we want to solve this problem, we need to go through the QUADRATIC PROGRAMMING.
The positive effect is that if a parameter has to go to 0, it will go 0 and this is useful because we want to simplify our model, we want to keep only the very essential measurements.

Q3. Logistic Regression

Q3.1 What are the main differences between the logistic-regression and the perceptron?

A3.1 PERCEPTRON is one of the most interesting classifiers which were developed under the Artificial Intelligence Framework.
We have a classification problem and we want to design a classifier, which is an hyperplane, through which we can split the points.
This algorithm works only under the linearly separable assumption, and it is implemented through only one neuron and a learning procedure which is not the descendent gradient procedure that we have seen also in the regression task.



We have only one neuron where we have the inputs, the parameters of the neuron and we have also an activation value. Then, the output is the Heaviside function, which says that:
- If the Activation Value < 0 the output of the neuron is 0.

- If the Activation Value > 0 the output of the neuron is 1.
- If the Activation Value = 0 the output of the neuron is 0.5.

As we said, the learning procedure is not the gradient descent procedure, but it's the following one, where we still have a dependence of the next one on the previous one and we have also a learning rate that is preceded by a minus because ideally, we would like to minimize the loss function.

$$w_{t+1}^j = w_t^j - \varepsilon_L (y_i - f(w_t, x_i)) x_i^j$$

However, perceptron was unable to solve trivial problems like the XOR problem, because this is non-linearly separable, we can't simply draw a line splitting the points.

Because of this, we can consider the same architecture, with more than one hidden neuron and the only thing we do is to substitute the output function from a linear function to a sigmoidal type of function, which is saturating, providing class 0 or 1.

If we apply this architecture, considering a sigmoidal function (which is differentiable), the non-linearly separable problem can be solved.

$$V_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\theta, x_i))$$

$$\hat{\theta} = \arg\min_{\theta \in \Theta} V_n(\theta) \; \theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}\Big|_{\theta=\theta_i}$$

By considering this neural architecture with a feed forward neural network with no linearity in the hidden layer and output layer, we are able to solve any classification problem.

However, when we look at the output of the last neuron, what we get is something close to 0 or close to 1, but this is not providing us any probability and this is one of the main differences with respect to what we can achieve with Logistic Regression.

We only know that we are on a certain part of the curve and if we are on the right side the output is 1 and if we are in the left side the output is 0, but we don't attach any probability.

With the **LOGISTIC REGRESSION**, we want to intend the output in probabilistic term, meaning that if we get a value that is for example in the right side of the sigmoidal function, we want this represents the probability of being in the class 1.

We are not only interested in saying the class is 0 or 1, but we want also to provide the probability because we want to provide a probabilistic interpretation for this architecture.

In particular, we have to intervene in the loss function, which is no ore the Mean Square Error, but the Cross Entropy.

After mathematical procedures, we can define the Likelihood, which is obtaining starting from the Canonical Form in which we can write the conditional probabilities, and then we can take the log likelihood, which has to be maximized in order to maximize the plausibility of being able to get a probabilistic interpretation.

$$Pr(y_1, y_2, \ldots, y_n | x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i} \qquad \ell(\theta) = \sum_{i=1}^{n} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

Q3.2 Discuss the major limit they share and how neural networks can solve it.

A3.2 The greater limit in perception and logistic regression is that in both cases a linear relationship is assumed between input and output variables.

As we have seen, for example, the perceptron is characterized by an output neuron with a Heaviside Function and it is able to work only through a linearly separable assumption, and we have seen the limit of this for example in the XOR problem, which can't be solved with this type of architecture.

This problem can be solved with Neural Networks where the difference is that we have nonlinear functions (such as the sigmoidal function), both at the level of hidden neurons and output neurons. In this way, there is the possibility to model more complex relationships between variables.

Q3.3 What is the role of activation functions in feedforward neural networks.

A3.3 Activation functions are fundamental components of feedforward neural networks and serve to introduce nonlinearities into neural networks, allowing complex models to be learned and represented.

In particular, activation functions are used both in hidden neurons and output neurons of neural networks:
→ in Hidden Neurons, activation functions introduce nonlinearity into the input data, allowing the network to learn complex representations of the data.
→ in Output Neurons, activation functions determine the final output of the neural network.
Each neuron has a certain activation value, which is calculated as a linear combination of its inputs and subsequently an activation function, which allows the output of the neuron itself to be determined as a function of the activation value.
We can say the activation value represents the raw input of the neuron, while the output is how the neuron contributes to the final output of the network.
Depending on the type of problem that we are facing (for example classification or regression), the activation function can vary and we have seen that, for example for the output neuron, we have a linear function in the case of regression, while we have a sigmoidal type of function in the case of classification.
In particular, in a binary classification problem, the activation function in the output neuron is a sigmoid, which returns a value between 0 and 1. The activation functions also fall within the scope of the Back Propagation Algorithm, which is the learning procedure that is used in the case of Neural Networks.

Q4. Consider the regression problem shown in the picture below and answer each point.
Q4.1 Do you think a model of the family f(x, theta) = $\theta_0+\theta_1 x_1+\theta_2^* x_2^*$ is a good choice for such task? Why?
A4.1 In general, we should use a linear model when we have some priors related to the problem. In particular, if we know that the Data Generating Process is reasonably linear, it may be ideal to use a linear model. Furthermore, it could be to use a linear model if we have a small data set from training and therefore a particularly high variance.
However, there are some cases where a linear model like the one proposed above is not sufficient, and this is the case of the situation represented in the figure.
The points follow a parabolic trend, which can't be approximated by a hyperplane that we could obtain through a linear model.
In this case, it's better to use a nonlinear model and we need to build the Loss Function, that is following exactly the same framework we were using also for linear regression:

$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y(x_i) - f(\theta, x_i))^2$$

We start from the very same setup and the only difference is that now the function f (x, theta) is a nonlinear function.
The way we follow is exactly the same, in the sense that we need to minimize the loss function in order to find the optimal parameters.
However, in this case we are not able to solve this problem in close form and the solution is the possibility to use Gradient-Based Optimization.

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}|_{\theta_i}$$

We know there are 2 variables, X and θ, and we want to minimize with respect to θ.

Q4.2 Do you think using a feed-forward neural network would improve the results?
A4.2 Considering the Structural Risk, we have seen that the major contribution seems to come from the Approximation Risk, which is related to the choice of the Family of Models.
Because of this, we have to understand which type of family of models we have to consider in order to solve our task if we don't know g(x).
In this situation, it could be useful to use a feedforward network, because as we know from a theorem that goes back to 1991, a feedforward neural network with a single hidden layer, containing a finite number of neurons and a linear output neuron can approximate any continuous function defined on a compact subset.
In this way, we can design a function with a finite number of neurons which is able to approximate the unknown concept g(x).

In this specific context, we might have a starting idea about the family of models that might be suitable for our problem and then we could take it as a starting point to see if it is able to effectively capture the relationship in the data. Consequently, we may not have to resort on a feed forward neural network, avoiding in this way computational costs.
However, compared to the family of models proposed, a neural network would certainly offer better results, since through nonlinear activation functions in the hidden neurons, there would be the possibility of having a more flexible model and certainly more capable of explaining the points of the graph, which don't follow a linear trend.