ML Problem set 1

Krishna N. Dindial

February 25, 2026

1 Introduction

For this problem set I used linnear regression, k nearest neighbors and multi-layer-perceptron to try to predict a star's gravity using the spectra of the star. I "scored" the model by calculating the difference between the model and the validation data squared. I then summed up the errors squared for each point in the validation set and used that sum. The units of this score are a bit messed up... it would

2 Linnear Regression

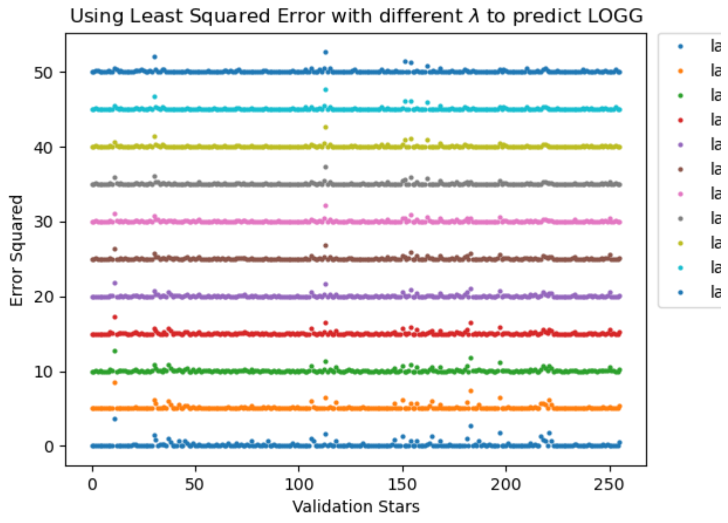
For the linnear regression I used np.linalg.solve() to solve the equation

$$Y^* = X^* X^T (XX^T)^{-1} Y \tag{1}$$

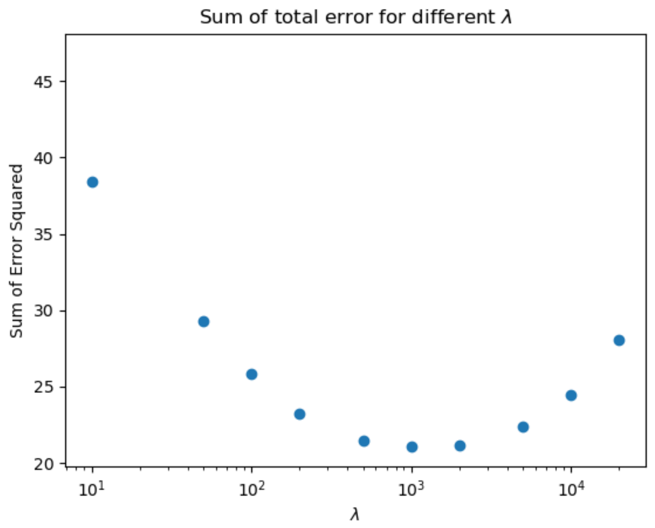
Where the design matrix made from X is the spectra of the training data, y is the surface of the of the training data, X* is the validation spectra design matrix, and Y* is the validation gravity. This equation works when the number of parameters is larger than the size of the training data. I wanted to play around with different parameters, so I added in a regularization factor λ, which changes the equation to

$$Y^* = X^* X^T (XX^T + \lambda I)^{-1} Y \tag{2}$$

To see how this affected my error, I plotted the error squared for each star as a function of different lambdas.



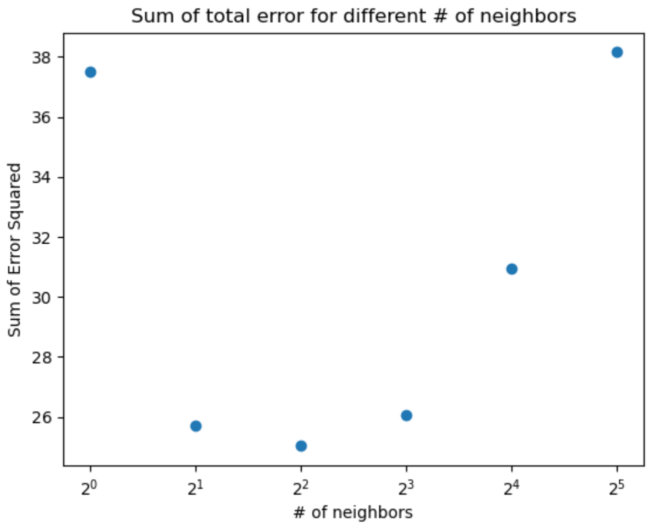
Its a bit of a strange plot, (I offset each lambda by 5 so that they would not all be stacked on top of eachother). But its interesting to see how for some stars with large error, the error gets smaller as lambda increases and for others the error gets larger as lambda increases.



To pick the best lambda, I plotted the total error squared as a function of lambda. The optimal lambda seems to be around 1,000, with a total error of 21. I will use this value of lambda on the test set in a later section where I compare the models to the test data

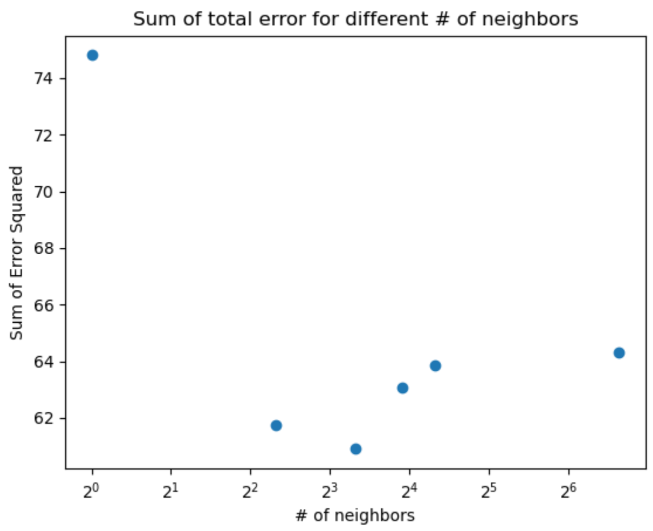
3 K Nearest Neighbors

We have around 8k paramters, so I was a bit initimidated to calculate an 8k dimensional distance, for each star in the training data and each star in the validation data. I asked chatGPT if there was a cleaner way to do this, and it gave me two ideas: first, it told me theres a sci kit learn algorithm that does this for you. Second, it suggested running a PCA to get the number of features down to a smaller number and then trying nearest neighbors. First I just tried running nearest neighbors for different neighbor sizes. I plotted error vs neighbor size for neighbor sizes 1,5,10,15,20,100.



KNN with 5 neighbors performed the best with a total squared error of around 25. 1 neighbors and 10 neighbors had similar performances. I will use the nearest neighbors model with 10 neighbors on test data, because having a larger number of neighbors is probably less sensitive to noise and outliers, but if the number of neighbors gets too large the performance drops.

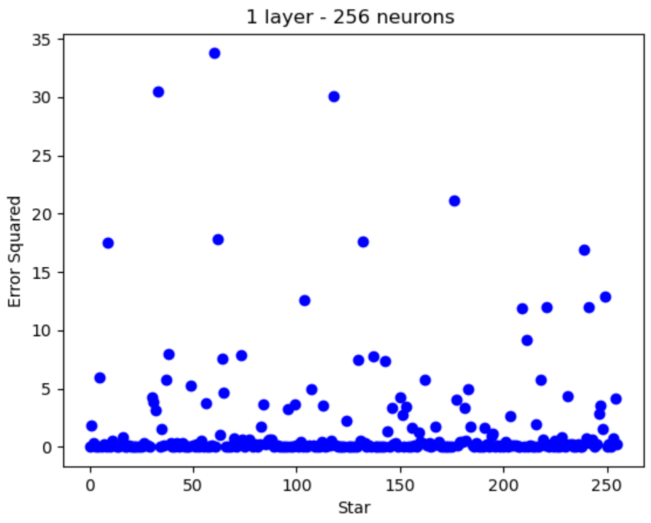
I then tried chatGPTs idea of ”taming” the parameters. I imported PCA from sci-kit-learn and reduced the number of parameters to 100 then ran K nearest neighbors.



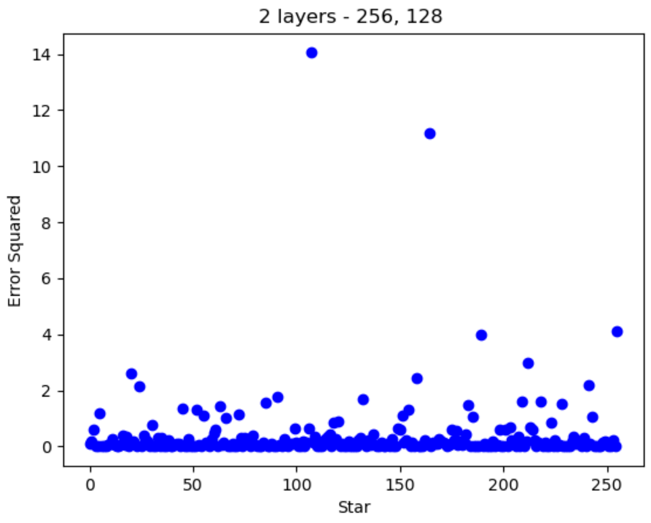
The squared error was around twice as large. Before the PCA including neighbors made the performance get worse, but after PCA including more neighbors seemed to have less of an affect on the performance. So obviously, compressing the data made the k nearest neighbors algorithm worse, but it was not clear how much worse this would make it. There seemed to be no benefit of doing the PCA. The nearest neighbors algorithm was fast and ran with no problems without compressing the number of parameters.

4 Multi-Layer Perceptron

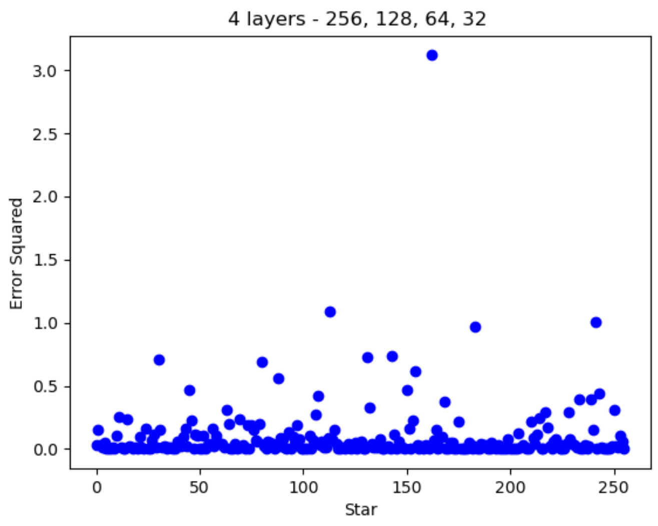
For the multi-layer perceptron model I used the MLPRegression function from sci-kit-learn. I used RELU as nonlinearity function. I used ADAM as my optimizer with an alpha parameter of 3e-4 per the suggestion of https://karpathy.github.io/2019/04/25/recipe/. For my first attempt I started with 256 neurons.



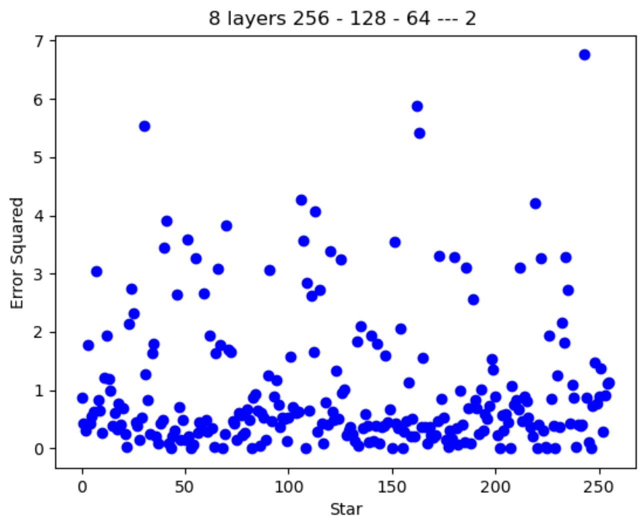
The sum of the squared error was: 466.4143956879076, so this is much worse than linnear regression and KNN. My next thought was to send it through a layer half the size, so for the second attempt, I passed the same parameters, except now I had two layers of size 256 and the second of size 128



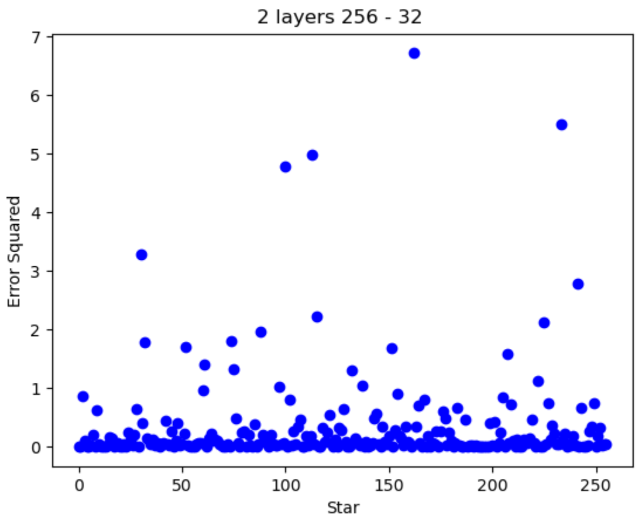
The sum of the errors was around 102. So this was an improvement. I then tried 4 layers, of size 256,128, 64 and 32. My motivation for this strategy is that I visualized the diagram hogg often draws of converging from a large number of parameters, down to a smaller number of parameters



The sum of the error was around 25 so this was comparable to KNN and linnear regression. I then decided, why not go all the way with this pattern and I did 8 layers, from 256 to 2 and each layer getting half as big



The total loss squared was around 256, so it did not out perform the 4 layer version of this pattern. The next thing I wanted to try was skipping the intermediate layers. I tried it with just two layers, going from 256 to 32.



The sum of the errors was 85. I tried even more combinations of layers, but this would get very redundant, so I am not including them all here, but they are in the notebook 'problem1.ipynb' and there are more plots in the figures section of the repository. The best scoring MLP I tried had 4 layers, going 256,128,64,32. I tried playing with different activation functions and alpha sizes but it did not change the performance much

4.1 Randomization

We are asked to run the same regression with different random seeds.Since the model with 4 layers decreasing in size from 256 to 32 had the best performance, I used the same set up with 5 different random seeds. The sum of squared error for the different random seeds were: 23.865751368205487, 25.654588186892315, 28.950418229217433 and 27.566260041461035. The performance of the model converges.

5 Performance on Test Data

I measured the performance by calculating the difference betweent the prediction of the LOGG of each planet in the test set and the actual LOGG of the test set, then squared this difference and then calculated the sum. Note: there are 512 stars in the test data and 256 stars in the validation data, so to compare these scores to the validation scores, you would divide by 2.

Using linnear regression with a regularization parameter of 1,000 the sum of the errors was 37.00580722200459.

Using KNN with 10 neighbors, the sum of the errors was 53.945824

Using MLP with layer sizes of 256,128,64 and32 the sum of the errors was 502017.9059077923. I way overfit my shit!! With MLP, I was trying to get the error to a comparable range as KNN and linnear regression, but in the process I overfit my data.

6 Final Project

My research group published this paper where they used machine learning to predict the mean free path of a InAs quantum well from the AFM data. https://arxiv.org/pdf/2409.17321 I think it would be interesting to see if I could improve the model. Each afm image has a crosshatch pattern, and before sending the afm data into the ML model, my colleagues, deconvolved the AFM image to extract the crosshatch pattern to de-noise the data. This was a slow, manual process that required rotating each afm image and as a result they really did not get enough data. I think I could make a model that skips this deconvolution step, and also improve the model One big problem I could see is that we might not have enough afm data. Each afm scan takes around 30 minutes and the data can be noisey, but if I can at least skip this manual de-noising process it will increase amount of data we have by a lot.

If I do not end up doing this for my final project, I would like to do something with sports gambling.