CS-UY 4563: Machine Learning

Final Project Written Report

Housing Price Prediction

December 7, 2022

Professor Linda N. Sellie

Jonathan Guan, Mark Deng

**Introduction**

This project examines the effectiveness of machine learning models for predicting the prices of housing, using a dataset of 1460 houses, and including 80 different features. The models used to fit the data include linear regression and deep learning neural networks. Support vector machines were also used on a modified version of the problem, which classifies each house as either above or below a $200,000 rather than an actual prediction on the house price. Each model was run using different regularizations and transformations, and tested with a varied set of hyperparameters.

**Pre-processing**

First the features were split between quantitative and categorical features. Then these categorical features were then split further, between features with some inherent order, and those without any order. These features include features such as basement quality, which has categories such as excellent, good, typical, fair, poor, and no basement. These features with inherent order were encoded and replaced with ordinal encoding, while other categorical features with no inherent order were encoded using one-hot encoding. After this processing, the number of features increased to 228.

Then, a second set of data was made by scaling and normalizing the data. This set of normalized data was to be used with SVM, as SVM requires scaled and normalized data to be effective, and the scaled data should also improve the gradient descent for the deep learning model.

The data was then split into a training set and a validation set, split 70% training and 30% validation. The split was done using a fixed randomization to ensure no bias in the order of the data.

**Linear Regression**

The first model is linear regression. Using a basic linear regression model without any regularization, the model scored an accuracy of 92.36% on the training set, and an accuracy of

87.74% on the validation set, which is quite a good result despite no regularization for overfitting. We also tried using both L1 and L2 regularization, with alpha values [0.001, 0.01, 0.1, 0.5, 1, 5, 10, 50 ,100, 500]. The same regularizations were performed on a linear regression model with a transformation by squaring the features, with alpha values [1 ,5, 10, 100, 500, 1000, 10000,50000,100000]. The results of the regularizations and transformation are all shown in the results below. Each graph shows the accuracy of the model vs. the logbase10 of the alpha value.

## Results

## Linear Regression

No Regularization:

Train\_score = 0.923565 ; Val\_score = 0.877412

With Regularization:

### 

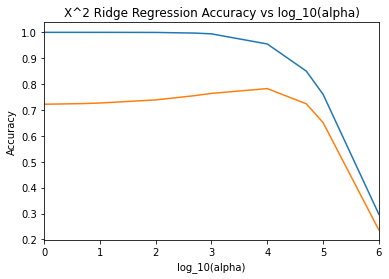
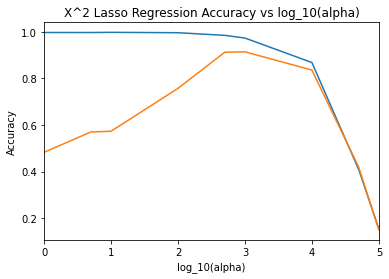
| Regulation  V.S. Alpha | 𝛼 = 0.001 | 𝛼 = 0.01 | 𝛼 = 0.1 | 𝛼 = 0.5 | 𝛼 = 1 | 𝛼 = 5 | 𝛼 = 10 | 𝛼 = 50 | 𝛼 = 100 | 𝛼 = 500 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X : L1 train | 0.915831 | 0.915832 | 0.915837 | 0.915856 | 0.916743 | 0.919504 | 0.921968 | 0.91658 | 0.908727 | 0.848883 |
| X : L1 val | 0.875113 | 0.875124 | 0.875235 | 0.875701 | 0.877519 | 0.88539 | 0.890373 | 0.89032 | 0.88811 | 0.862521 |
| X : L2 train | 0.923565 | 0.923551 | 0.922641 | 0.916169 | 0.910508 | 0.894387 | 0.886582 | 0.864746 | 0.853916 | 0.826374 |
| X : L2 val | 0.877445 | 0.877716 | 0.878716 | 0.874864 | 0.871274 | 0.867062 | 0.866935 | 0.865777 | 0.863918 | 0.855185 |

Polynomial Transformation - X^2

Without Regularization:

Train\_score = 1.0 ; Val\_score = 0.720595

With Regularization:

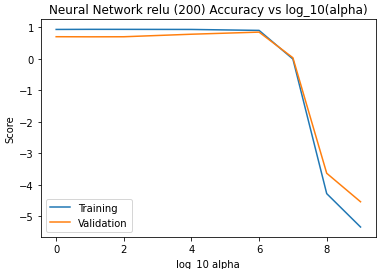


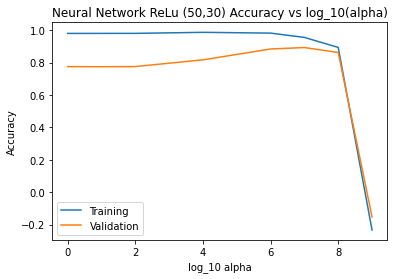
| Poly Transform V.S. Alpha | 𝛼 = 1 | 𝛼 = 5 | 𝛼 = 10 | 𝛼 = 100 | 𝛼 = 500 | 𝛼 = 1e3 | 𝛼 = 1e4 | 𝛼 = 5e4 | 𝛼 = 1e5 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X^2: L1 train | 0.997535 | 0.99746 | 0.998259 | 0.996643 | 0.985309 | 0.973756 | 0.868501 | 0.408588 | 0.153319 |
| X^2: L1 val | 0.48262 | 0.570107 | 0.573197 | 0.757422 | 0.912828 | 0.914504 | 0.836078 | 0.41872 | 0.148696 |
| X^2: L2 train | 1.0 | 0.999996 | 0.999989 | 0.999626 | 0.997159 | 0.99417 | 0.99417 | 0.850338 | 0.760614 |
| X^2: L2 val | 0.72228 | 0.72478 | 0.726963 | 0.738958 | 0.755415 | 0.764145 | 0.782706 | 0.723882 | 0.651456 |

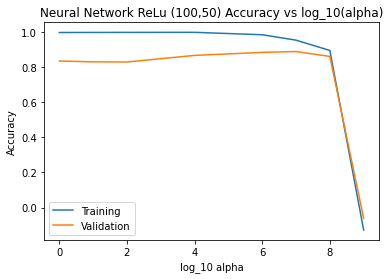
**Neural Network**

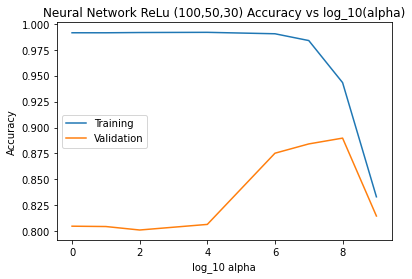
The second model we used was multiple neural networks with different layer structures. The hidden layer neuron structures we tested include (200), (50 , 30),(100, 50) and (100,50,30). We used sklearn’s MLPRegressor class to build the neural network. While running the neural network model, we found that when running the model with tanh and logistic activation functions on the hidden layers, the model took a much larger time to train, and the score of the resulting models only yielded negative results. In light of these findings, we decided not to proceed further with testing them, and instead focus only on ReLu. All neural networks models were implemented with Ridge Regression(L2) regularization with a list of alphas: [𝛼 = 1, 1e1, 1e2, 1e4, 1e6, 1e7, 1e8, 1e9] and tested while keeping the learning rate = 0.001 and max iteration = 5000 constant. Graphs of the accuracies of the different hidden layers vs. the logbase10 of the alpha values are presented below.

## Results









## Neural Network (L2)

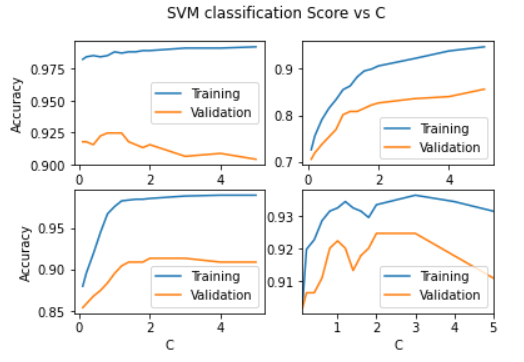
| Layer V.S. Alpha | 𝛼 = 1 | 𝛼 = 1e1 | 𝛼 = 1e2 | 𝛼 = 1e4 | 𝛼 = 1e6 | 𝛼 = 1e7 | 𝛼 = 1e8 | 𝛼 = 1e9 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 layer (200) ReLu train | 0.930685 | 0.933757 | 0.933134 | 0.930365 | 0.900219 | -0.008665 | -4.281182 | -5.340695 |
| 1 layer (200) ReLu val | 0.69965 | 0.696505 | 0.698348 | 0.778703 | 0.845374 | 0.031893 | -3.634808 | -4.539521 |
| 2 layer (50,30) ReLu train | 0.97974 | 0.979989 | 0.980098 | 0.986454 | 0.981692 | 0.955111 | 0.893454 | -0.233201 |
| 2 layer (50,30) ReLu val | 0.774856 | 0.77438 | 0.775214 | 0.816664 | 0.883672 | 0.892577 | 0.861989 | -0.152732 |
| 2 layer (100,50) ReLu train | 0.996272 | 0.996644 | 0.996885 | 0.997213 | 0.983514 | 0.952748 | 0.89406 | -0.128005 |
| 2 layer (100,50) ReLu val | 0.833965 | 0.829344 | 0.828324 | 0.865951 | 0.883122 | 0.887921 | 0.860019 | -0.061769 |
| 3 layer ReLu train | 0.991405 | 0.991396 | 0.991643 | 0.991818 | 0.990442 | 0.983891 | 0.943326 | 0.833184 |
| 3 layer ReLu train | 0.804836 | 0.804515 | 0.801156 | 0.806559 | 0.875232 | 0.884191 | 0 .889789 | 0.814665 |

**Support Vector Machines**

Finally we used support vector machines to do classification on the data. Due to the predicted variable, predicted house prices, being a continuous and not categorical variable, we decided to instead do prediction on whether or not houses are above or below a $200,000 value, turning the regression problem into a classification one. The data used for training was also scaled and normalized

We chose to use 4 different kernel functions for SVM, linear, polynomial of degree 3, radial basis function, and sigmoid. Each model was run with L22 normalization. Each model was run on multiple different values of the regularization term, C, and the accuracies of each model were plotted against the value of C in the figure below.

## Results



Linear (Top Left) ; Poly - degree = 3 (Top Right); RBF (Bottom Left); Sigmoid (Bottom Right)

## SVM (L2^2)

| Kernel V.S. C | C = 0.1 | C = 0.2 | C = 0.4 | C = 0.6 | C = 0.8 | C = 1 | C = 1.2 | C = 1.4 | C = 1.6 | C = 1.8 | C = 2 | C = 3 | C = 4 | C = 5 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Linear train | 0.982387 | 0.984344 | 0.985323 | 0.984344 | 0.985323 | 0.988258 | 0.98728 | 0.988258 | 0.988258 | 0.989237 | 0.989237 | 0.991194 | 0.991194 | 0.992172 |
| Linear val | 0.917808 | 0.917808 | 0.915525 | 0.922374 | 0.924658 | 0.924658 | 0.924658 | 0.917808 | 0.915525 | 0.913242 | 0.915525 | 0.906393 | 0.908676 | 0.90411 |
| Poly-X^3 train | 0.726027 | 0.755382 | 0.790607 | 0.815068 | 0.833659 | 0.855186 | 0.863014 | 0.882583 | 0.895303 | 0.899217 | 0.906067 | 0.921722 | 0.938356 | 0.947162 |
| Poly-X^3val | 0.705479 | 0.719178 | 0.737443 | 0.753425 | 0.769406 | 0.80137 | 0.808219 | 0.808219 | 0.815068 | 0.821918 | 0.826484 | 0.835616 | 0.840183 | 0.856164 |
| RBF train | 0.879648 | 0.895303 | 0.918787 | 0.944227 | 0.966732 | 0.975538 | 0.982387 | 0.983366 | 0.984344 | 0.984344 | 0.985323 | 0.988258 | 0.989237 | 0.989237 |
| RBF val | 0.853881 | 0.858447 | 0.86758 | 0.874429 | 0.883562 | 0.894977 | 0.90411 | 0.908676 | 0.908676 | 0.908676 | 0.913242 | 0.913242 | 0.908676 | 0.908676 |
| Sigmoid train | 0.903131 | 0.919765 | 0.922701 | 0.928571 | 0.931507 | 0.932485 | 0.934442 | 0.932485 | 0.931507 | 0.92955 | 0.933464 | 0.936399 | 0.934442 | 0.931507 |
| Sigmoid val | 0.901826 | 0.906393 | 0.906393 | 0.910959 | 0.920091 | 0.922374 | 0.920091 | 0.913242 | 0.917808 | 0.920091 | 0.924658 | 0.924658 | 0.917808 | 0.910959 |

**Analysis and Conclusion**

We analyzed each model’s effectiveness based on their best performance on the validation set. We use the validation set to avoid training bias, due to the tendency for models to overfit to the training data, making the model less generalizable. By evaluating on the validation set, we lose the training bias and get a better estimate of the true performance of the model on real world data. In all of our graphs, we plot the score of each model on the training and validation set, as it compares to the regularization hyperparameter that is relevant to the model. In every model, we expect a similar pattern - as the regularization is low, the model will tend to overfit to the noise of the training data, causing it to perform very well on the training set, but perform badly on the validation set. As the regularization term increases, we expect to see the performance on the validation set to increase up to a point. After hitting the peak, we expect to see the performance on the validation set to decrease, as the regularization term overpowers the training, and the model starts to underfit the data. In contrast to the performance on the validation set, we expect the performance of the model on the training set to continuously decrease as the regularization term is increased, due to the tradeoff between training bias and variance. After running all of the aforementioned models, we see that this trend is generally observed in every one of the models used.

The best model from the linear regression models was Lasso regularization on the X2 polynomial transformation and an alpha value of 1000, performing with an accuracy of 91.4504% when run on the validation set. While this model performs better than the other linear regression models, the difference between them is rather small. Lasso regularization on the linear model performed with 89.0373% accuracy when run on the validation set, which trails behind X2 transformation by only around 2.4%. The improved accuracy on the X2 transformation model suggests that the housing price may not purely be linearly related to the features, but due to the small difference between X and X2 models, this improvement may simply be just attributed to randomness in the data, and a strong conclusion can’t be made.

For the neural network models, as concluded earlier, the tanh and logistic activation functions yielded only negative results, approximately around 0. An attempt to resolve this issue was made by trying to scale the data using sklearn’s library, StandardScaler, along with an attempt to normalize the anticipated output (y=SalePrice) as a log normal distribution was observed from the distribution of the sales data, but we still received negative score results. However, the neural network using a ReLU activation function and a network structure of (50, 30) performed well with an accuracy of 89.2577% on the validation set, and alpha = 10,000,000. We noticed that as we add more layers to the network, the alpha values were required to be much larger to achieve reasonable scores. This is also consistent with the idea of overfitting as it relates to layers, as more layers adds to model complexity, and thus increases the amount of overfitting. Thus, a larger regularization term is required to prevent this overfitting. Overall, the neural network model performs well, and is comparable to the simple linear regression model in terms of performance.

For SVMs, the best score achieved was 92.4658% accuracy, which was shared by both the linear kernel SVM and the sigmoid kernel SVM. Our graphs showing SVM for polynomial kernel of degree 3 and RBF kernel are shown to be still increasing with C. However, we have tested both with higher C values, which have shown that the validation does begin to decrease, but only at C values of around 1000. The largest accuracy scores on the validation for both RBF and degree 3 polynomial with larger values of C however are still less than the accuracy scores achieved with linear and sigmoid kernels.

Among all models used in this project the best performance for predicting house pricing coming from the X2 transformation linear regression model, while the best SVM model for predicting the price above or below $200,000 is tied between linear kernel SVM and sigmoid kernel SVM. However, the all of the models were able to score well, with each model scoring at least a 78% accuracy, and typically lying between the 80 - 90% prediction accuracy range on the validation set. The success of these models show that we can predict house prices to a good degree of accuracy using the 80 features of the house that we analyzed. Despite the success, we have only provided a very simple look at the performance of machine learning models on house pricing - to find a better performance, we’ve identified a few changes that could be made. In regards to linear regression, there are more transformations on the data could be tested. For neural networks, there are many network neuron structures that may perform better than our current configuration, and activation functions such as leaky ReLU and ELU, that we have not tested in this project. In regards to SVM, there may be other kernel transformations to be tested. Besides the models listed, there are many other machine learning models for regression and classification that may perform better on the data than the ones shown in this project. And finally, having more training data, or more features may also improve the accuracy of the models.

**Works Cited**

*1. supervised learning*. scikit. (n.d.). Retrieved December 7, 2022, from https://scikit-learn.org/stable/supervised\_learning.html

Brownlee, J. (2020, August 17). *Ordinal and one-hot encodings for Categorical Data*. MachineLearningMastery.com. Retrieved December 7, 2022, from https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

Dgawlik. (2017, February 20). *House Prices Eda*. Kaggle. Retrieved December 7, 2022, from https://www.kaggle.com/code/dgawlik/house-prices-eda/notebook#Overview

*House prices - advanced regression techniques*. Kaggle. (n.d.). Retrieved December 7, 2022, from https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

John, R. T. (2022, June 26). *Regularization of linear models with SKLearn*. Medium. Retrieved December 7, 2022, from https://medium.com/coinmonks/regularization-of-linear-models-with-sklearn-f88633a93a2

Skashperova. (2022, December 1). *House prices beginners*. Kaggle. Retrieved December 7, 2022, from https://www.kaggle.com/code/skashperova/house-prices-beginners