Final Project Report

HOUSE PRICES PREDICTION(Using Boston Dataset)

Poojitha Mutteneni - UCF
poojitha.mutteneni@ucf.edu
5490225
CECS

1. Introduction

In this project, I used various Machine Learning Techniques which were taught and implemented in class. For this final project, I focus on comparing performance of these different models for house prices prediction. All the way from simple linear regression, examining performance of polynomial regression models with different degrees to neural network models, and evaluating them using **cross-validation**. House prices prediction is an essential aspect, as we have seen clearly in recent times during the end of COVID we saw a buyers market and during recession we saw different patterns, it's imperative we understand the underlying factors to predict.

The dataset used was the Boston Housing dataset which has **506 entries**, each representing data from a neighborhood. It comes with **13 features**, such as (CRIM), (NOX), (RM), and so on. The target variable (**MEDV**) is the median value of homes in thousands of dollars.

The heatmap shows the relationship between MEDV and the features. RM is positively correlated (0.70), while LSTAT and PTRATIO are negatively correlated (-0.74 and -0.51, respectively). Strong correlations between features like TAX and RAD (0.91) help understand possible multicollinearity.

Initially, polynomial regression with an optimal degree of 2 was applied, achieving impressive training and testing performance with a high R2 of score of 0.93 and 0.79, respectively. The model demonstrated well on unseen data. For task 3, I implemented several neural network models, and model 3 achieved an average cross-validation performance with a slightly higher R2 score of 0.85 and mean squared error (MSE) of 11.8 on the test set.

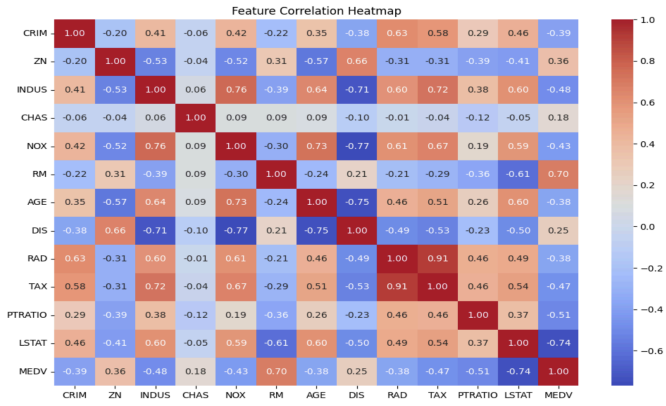

Fig: Heatmap for feature comparison

Once data was loaded, I checked its structure and summary statistics to get a sense of the distributions and ranges of each variable. Importantly, I confirmed that the data had **no missing values**.

I split features and target taking the first 12 columns as X and the last column (MEDV) as y. Next, I also split the data into **80-20** ratios as training and testing seemed to be the best choice for the size of dataset used for this project.

The Boston dataset has features on a range of different scales, like crime rate (a decimal) and tax rate (a few hundred). Therefore I standardized the data using StandardScaler to prevent any bias.

For all my models I used the evaluation metrics—Mean Absolute Error, Mean Squared Error , Root Mean Squared Error, and R^2(shows prediction power).

MAE: Measures the average absolute difference between actual and predicted values.

$$MAE = \frac{\sum_{i=1}^{n} |y_{pred,i} - y_i|}{n}$$

MSE: It is the average squared error between actual and predicted values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

RMSE: Represents the square root of MSE

R²: Represents the portion of variance the model explains seen in target.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

2. Task 1 - Regression

2.1 Simple Linear Regression

To start my initial experiments, I used simple linear regression. It can be defined as a model in which we try to establish the **relationship** between an independent variable X and a dependent variable Y. In an equation it can be written as,

$$\boldsymbol{Y = aX + b}$$

The relationship between them is understood by fitting a line between them.

It is not necessary that the values always will be true, there may be slight differences in the value, so we maintain an error term.

$$\boldsymbol{Y = aX + b + E}$$

Here Y = Predicted variable, a = Slope, X = independent variable , E = error term.

I trained the model on the scaled training set and predicted housing prices for both training (Ein) and testing (Eout) sets.

The results were:

Ein (Training): MAE: 3.42, MSE: 22.60, RMSE: 4.75, R²: 0.74
Eout (Testing): MAE: 3.11, MSE: 22.78, RMSE: 4.77, R²: 0.69

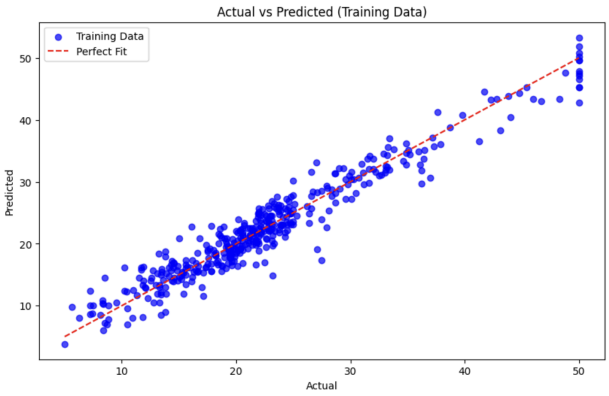

Fig: Actual vs Predicted Labels - Training Data

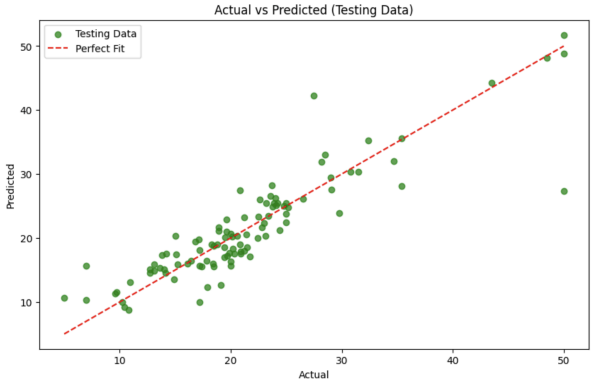

Fig: Actual vs Predicted Labels - Testing Data

2.2 Polynomial Regression

Linear models alone might not capture complex relationships entirely. In order to take into consideration nonlinear **relationships** in the data, I moved to polynomial regression.

It can be known in an equation format as,

$$\boldsymbol{Y = a + b1X1 + b2X2 + ... + bnXn}$$

First, I evaluated **degrees from 1 to 5 using cross-validation** and selected the one with the highest average R² score. This turned out to be **degree 2**.

After creating second-degree polynomial features, I split the data again and trained the model .Polynomial regression models are more prone to overfitting due to their higher complexity. By splitting the data after creating polynomial features we can test the model's generalization with the same complexity. The results showcased better values compared to simple linear regression:

Ein (Training): MAE: 1.84, MSE: 5.72, RMSE: 2.39, R^2: 0.93
Eout (Testing): MAE: 2.56, MSE: 15.15, RMSE: 3.89, R^2: 0.79

Thereby the results showcase the advantages of the model over simple linear regression; it was able to capture the nonlinear relationships, by selecting the right degree. It is to be checked as it may cause overfitting with larger values.

2.3 Ridge Regression

I further tried to evaluate if the model was going through overfitting therefore I attempted to use Ridge regression. Ridge adds a **penalty** or Bias to large coefficients, helping the model generalize better. Thereby reducing the overall complexity of the model, also termed as L2 regression. The calculated Bias is added as Lambda*

I tested various alpha values, ranging from 10-6 to 106 using 10-fold cross-validation. The **optimal alpha** was found to be 10-6 , which achieved a strong balance between bias and variance.

With the Ridge model:

Ein (Training): MAE: 1.84, MSE: 5.72, RMSE: 2.39, R^2: 0.93
Eout (Testing): MAE: 2.56, MSE: 15.15, RMSE: 3.89, R^2: 0.79

Interestingly, the results remained consistent with the unregularized polynomial model, suggesting that overfitting wasn't a significant issue at this stage.

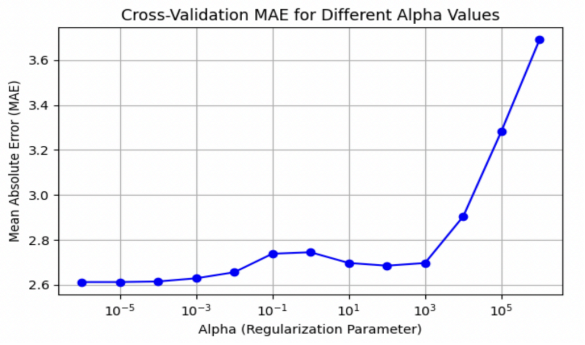

Fig : Optimal Alpha Value for Ridge Regression

3. Task 2 - kNN

In the KNN algorithm, the number of neighbors (k) helps us to determine the quality of our predictions thereby playing an important role. To identify the best value for k, I explored a range of k values from 1 to 30. For each value of k, a KNN regression model was constructed and evaluated using 10-fold cross-validation. Where data is split into 10 sets and trained on nine of them and tested on the other one.

Once the **Ecv** values for all tested k values were scored then the next step I determined was the optimal k. This was done by identifying the k value which aligns to the smallest cross-validation error. This optimal k value was then used to train a new KNN model.

I plotted a visual representation of the relationship between k values and cross-validation errors. This graph shows **optimal k**. The red line in the figure helps us analyze this. According to the graph the optimal k value is 3. It also helps knowing errors increased with excessively low or high k values and understanding overfitting or underfitting in the data.

I believe kNN did not perform better compared to polynomial regression due to the size of the dataset being small, hence effectively estimating local neighborhoods was a challenge. Polynomial regression captures complex, non-linear relationships globally compared to kNN relies solely on local averaging.

KNN Scores:

Training - (E_in):MAE: 1.893, MSE:8.58, RMSE: 2.93, R^2 Score: 0.90
Testing - (E_out): MAE: 2.64, MSE: 18.72, RMSE: 4.32, R^2 Score: 0.74

k	Ecv
1	25.08
2	21.34
3	**20.93**
4	23.32
5	24.93
6	25.94
7	25.66

8	26.60
9	26.46
10	26.28

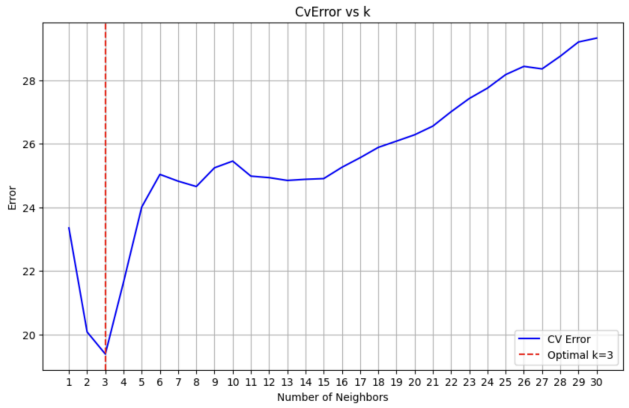

Fig : Cv Error vs K

4. Task 3- Neural Network

As a final model, I explored neural networks to predict housing prices based on the Boston dataset. Neural networks are highly flexible models that can capture complex, non-linear relationships in data. For this task, I tried experiments with different network architecture to analyze how they impact the performance.

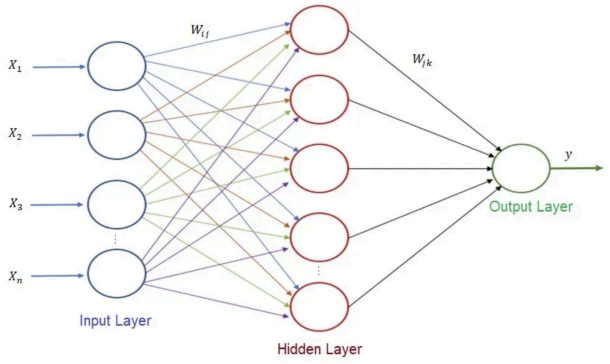

Fig: Neural Network Structure

I used five models. The architectures included some changes in the number of layers, activation functions, or dropout rates.

Models Architecture Overview

Model 1: Input Layer → Fully Connected (64 units) → Tanh Activation → Fully Connected (1 unit, output layer).

Model 2: Input Layer → Fully Connected (128 units) → ReLU Activation → Dropout (30%) → Fully ,Connected (1 unit, output layer).

Model 3: Input Layer → Fully Connected (128 units) → Tanh Activation → Fully Connected (64 units) → Tanh Activation → Fully Connected (1 unit, output layer).

Model 4: Input Layer → Fully Connected (256 units) → ReLU Activation → Fully Connected (128 units) → ReLU Activation Fully Connected (64 units) → ReLU Activation → Fully Connected (1 unit, output layer).

Model 5: Input Layer → Fully Connected (128 units) → ReLU Activation → Dropout (40%) → Fully Connected (64 units) → ReLU Activation → Fully Connected (1 unit, output layer).

Models were trained using the **Adam optimizer** (a variant of **SGD**), with a **learning rate of 0.005**, along with a **step based** learning rate scheduler to adapt to the learning rate over time. Each model I trained for a maximum of 300 epochs with a batch size of 32. **Early stopping** was implemented which was explained in class to terminate training if the validation loss was not improving over the epochs to prevent overtraining. In the code, I used **backpropagation** which calculates loss function of w.r.t Models parameters. I also used **K-Fold Cross Validation**, although I first tried 5 fold, I felt the model performed better when stopped at 2 fold and continued as such.

Below shows a table to compare how different architectures performed.

Model	MAE	MSE	RMSE	R2
M1 - E_in	1.5654	5.0792	2.2208	0.9389
M2 - E_in	2.1148	8.5501	2.9217	0.8982
M3 - E_in	1.0872	2.3163	1.5162	0.9727
M4 - E_in	1.25	2.9052	1.7031	0.9654
M5 - E_in	2.0011	7.7273	2.775	0.9079

Model	MAE	MSE	RMSE	R2
M1	2.6168	14.0999	3.7371	0.8314
M2	2.496	12.7583	3.5628	0.8477
M3	**2.3538**	**11.8154**	**3.4158**	**0.8586**
M4	2.6379	14.2757	3.7627	0.8294
M5	2.6231	14.1255	3.7433	0.8312

The results show that **Model 3** achieves the best performance. The lectures do emphasize also to adjust learning rate based on depth which could have given better results. Simpler models like M1 underfit the data, while deeper models like M4 and M5 may overfit or face vanishing gradients due to their depth.

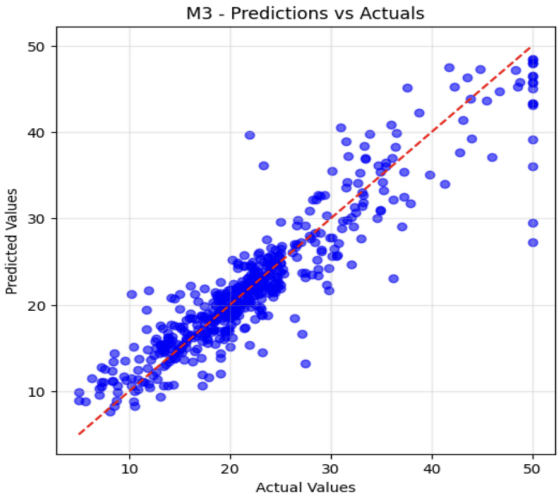

Fig: Model 3 Predicted vs Actual

5. Final Model Comparisons

Across all tasks, the performance of models for house price prediction is mostly a progression over the different models I used. Task 1 began with regression models, where Simple Linear Regression served as a baseline, achieving a test R^2 of 0.69 and RMSE of 4.77. Polynomial Regression (Degree 2) improved significantly by modeling non-linear relationships, yielding an R^2 of 0.79 and RMSE of 3.89, with Ridge Regression confirming minimal overfitting. Task 2 explored kNN Regression, which performed better than regression models with an R^2 of 0.74 and RMSE of 4.32 at an optimal k=3. Task 3 introduced Neural Networks, where Model 3, featuring two hidden layers (128 and 64 units) and Tanh activations, achieved the good performance with a test R^2 of 0.86 and RMSE of 3.41

For comparison among all overall Simple Linear regression was easy to implement and served as an initial baseline model and computationally efficient, but can't capture any non linear relationships and making it have lower R2 score showing lower predicting power and sensitivity to outliers. When taking into consideration non linear relationships, polynomial regression performed well, improved accuracy, but it's essential to choose the right degree as it might cause overfitting with high degree and increases model complexity. Ridge regression is usually supposed to improve on the overfitting issues if any, but in our case our results did not change and it is a bit more complex to implement and requires tuning alpha parameters. kNN is also easy to implement, captures local data patterns and is flexible. It is highly sensitive to the value of K selected and has higher computational costs compared to scalability. The predictive power increased overall, but polynomial regression played a better turn out in this data, but it performed better than simple linear regression. Finally, Neural Networks are used to understand more complex non linear and are adaptable to different data. All the structures of Neural Networks experimented formed better than the previous models, out of which the model 3's architecture gave the best results. Although it has high computational overhead and requires tuning of hyperparameters. It performed the best among the models for predicting boston house prices.

Model	MAE (Test)	MSE (Test)	RMSE (Test)	R^2 (Test)
Simple Linear Regression	3.11	22.78	4.77	0.69
Polynomial Regression (D=2)	2.56	15.15	3.89	0.79
Ridge Regression	2.56	15.15	3.89	0.79
kNN Regression (k=3)	2.64	18.72	4.32	0.74
Neural Network (Model 3)	**2.35**	**11.82**	**3.42**	**0.86**

Github Code Link : https://github.com/poojithamutteneni?tab=repositories

Video Link https://drive.google.com/file/d/1vbzUdjaEhxxQbYUbrMbLPqe3UCDV8nWq/view?usp=sharing