Final Project: Multi-task Learning for Predicting House Prices and House Category

Presented to
Professor Fatih Nayebi
TA Necmiye Genc

By
Delisle, Audrey - 261142504

MGSC 673 - Section 075

McGill University - Desautels Faculty of Management
Sunday April 28th 2024

Table of Contents

Section 1. Introduction

1.1 Project summary

This report details the development and evaluation of a multi-task learning model designed to predict house prices and classify house categories. The project uses the House Prices - Advanced Regression Techniques Dataset, with an added 'House Category' variable created from house features. A model architecture that shares a common bottom layer while employing task-specific top layers is developed using PyTorch Lightning. By creating the model in this sense, it leverages the efficiencies of multi-task learning but also demonstrates how different tasks can benefit from shared representations.

1.2 Project objectives

The primary objective of this project is to evaluate a multi-task learning model capable of handling two distinct tasks: predicting house prices, a regression task, and categorizing houses based on predefined criteria, a classification task. This involves creating a neural network that integrates these tasks to optimize learning and performance efficiency. Additionally, the project aims to explore the impacts of various activation functions, optimizers, and loss functions on the model's performance, ensuring precision and accuracy. Finally, the implementation should demonstrate the capabilities of PyTorch Lightning in managing complex machine learning workflows, from model training to hyperparameter tuning.

Section 2. Data Exploration and Preparation

2.1 Description of the dataset and its source

The dataset used in this project is from the "House Prices - Advanced Regression Techniques" competition on Kaggle. This dataset contains various features related to residential homes in Ames, Iowa, and includes both training and testing sets. The dataset features a wide range of variables, totaling 81, which describe almost every aspect of residential homes. These include basic property characteristics like lot size and zoning, details on the style and quality of the house, type of utilities available, and the presence of various amenities such as garages and pools. The target variable for the regression task, SalePrice, represents the property's sale price in dollars. For the classification task, additional features will be engineered to categorize houses based on a combination of existing attributes.

2.2 Steps taken for data cleaning

As mentioned above, this was a very big dataset. Hence, I did a lot of EDA to figure out which were the best variables to keep in the model and how to clean them. One thing I noticed right away was that there were multiple attributes specifically detailing one feature, such as garage features, leading to redundancy in the dataset. Hence, I removed a lot of the redundant columns.

Additionally, a good portion of the characteristics were categorical, posing challenges in terms of managing dimensionality and multicollinearity. To address these issues and streamline the dataset for subsequent analysis, I only kept the columns I found to be the most valuable. Then I did some data cleaning steps. Initially, missing values within both numerical and categorical variables were addressed. Categorical variables lacking information were uniformly assigned the label "None," while numerical variables were imputed with the median value of their respective columns. Then, I had to one-hot encode the categorical variables to be able to use them in my model. Lastly, I checked for duplicates in the data but everything looked fine.

2.3 Feature engineering

For the classification task, I needed to make a new category called "House Category." This category combines the variables year built, year remodeled, house style and building type. The process began with simplifying the house styles and building types to help with interpretability. This involved standardizing the names of various house styles and categorizing building types into broader classifications. The categories were 1.5 Story, 2PlusStory, Residential, and MultiFamily. Then, I leveraged the information from the 'YearBuilt' and 'YearRemodAdd' variables to determine the age category of each house. Specifically, if a house was constructed or remodeled after 1980, it was classified as "New"; otherwise, it was categorized as "Old." After that, I put all this information together into one category for each house. But I had to be careful not to have too many categories because that can make things complicated. So, I made sure I didn't have more than 15 categories for the classification task. This way, I kept my dataset easy to work with while still having all the important information I needed for my project.

2.4 Data splitting strategy (training, validation, testing)

The dataset was initially divided into two subsets: training and testing. This division was executed using the train_test_split function from the scikit-learn library, with a test size of 20% and a random seed set to 13 for reproducibility. Following the division, preprocessing steps were applied to both subsets to prepare them for modeling. The preprocessing steps comprised two primary stages: imputation and transformation. Numerical variables were imputed using the median value of each respective column, while categorical variables were imputed with a constant placeholder value ('Missing'). Subsequently, the preprocessed data underwent transformation to standardize numerical features and encode categorical variables. This transformation was facilitated by the ColumnTransformer, which helped with the application of distinct preprocessing steps to numerical and categorical variables separately. Specifically, numerical features were standardized using a RobustScaler, while categorical features were encoded using a OneHotEncoder.

Upon completion of preprocessing, the data was ready for model training and evaluation. The feature matrices (X_train and X_test) and target variables (y_train_price, y_test_price, y_train_category, and y_test_category) were prepared for input into the subsequent modeling pipelines. The y price variables were the house sales price and the y category were the house category variables after it got label encoded.

Section 3. Model Architecture and Design

3.1 Description of the model architecture

In order to predict house prices and categories simultaneously, I used PyTorch Lightning to construct a feed-forward neural network model. This neural network faces the dual challenge of handling both regression, for predicting house prices, and classification, for categorizing houses. At the core of the model lies a shared bottom model, which is called the feature extraction backbone. This backbone consists of carefully designed layers that extract essential features from the input data. These features enable the model to identify intricate patterns and relationships within the housing dataset.

To introduce nonlinearity and capture complex data structures, I used Rectified Linear Units (ReLUs) and Leaky ReLUs as activation functions. These functions help the model approximate highly nonlinear relationships effectively, enhancing its predictive capabilities.

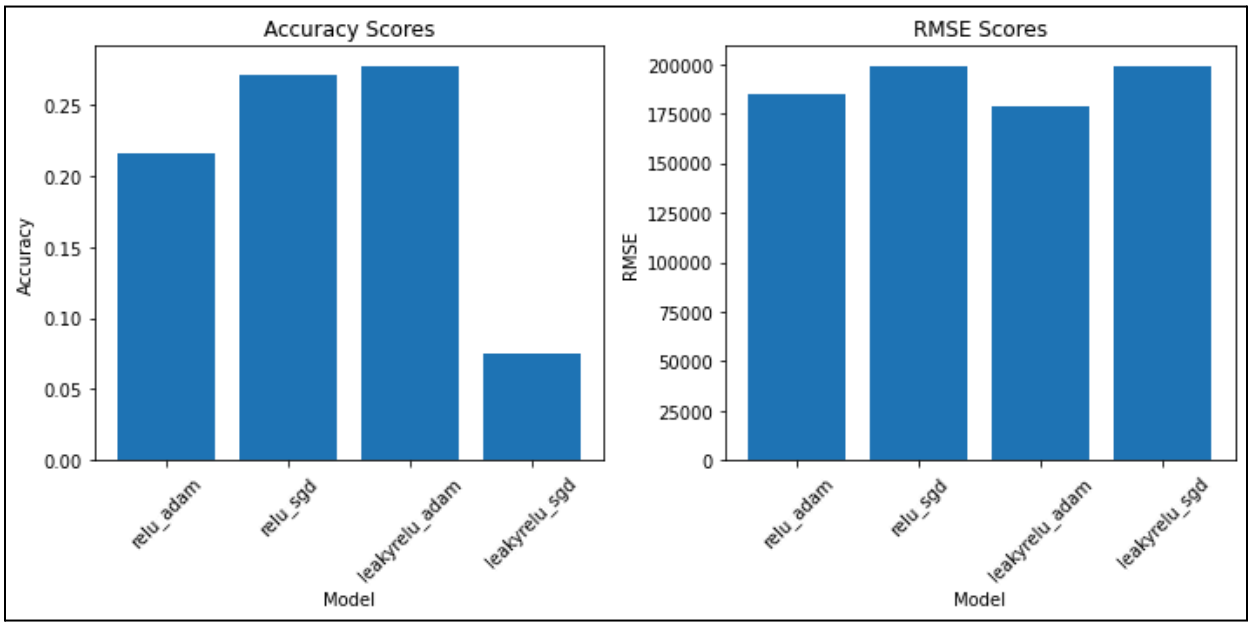
In addition to the shared backbone are task-specific top layers, namely the regression and classification heads. These layers are tailored to address the distinct objectives of predicting house prices and categorizing houses. The regression head, comprising a single neuron layer, is excellent at forecasting house prices accurately. In contrast, the classification head, with multiple neurons, is adept at assigning input instances to their respective categories with precision.

During training, the model undergoes optimization to refine its parameters and improve predictive performance. To achieve this, I employ a weighted combination of loss functions, ensuring attention to both regression and classification tasks. By balancing the mean squared error loss for regression and the loss for classification, the model aims to optimize its performance across both tasks.

3.2 Activation function and Optimizer exploration

In this phase, I am exploring different ways to improve our model through various activation functions and optimizers. Activation functions are like switches in the model that help it learn complex patterns in the data. I decided to test two types: ReLU and Leaky ReLU. ReLU is simple and effective, while Leaky ReLU helps deal with some tricky problems that can slow down learning. Next up are optimizers. These are like the engines that drive the model's learning process. I decided to compare two popular ones: Adam and SGD. Adam is smart and adjusts how fast the model learns as it goes along. SGD is more straightforward, just following the gradients to improve. In order to test both these parameters, I loop my model code over the different parameters and print out the accuracy (classification task) and rmse (regression task) each time. I then plotted the results in graph 1 (view below). After looking at the results, I can tell that the models using Leaky ReLU and Adam tend to do the best. They are very good at classifying house categories correctly and predicting house prices accurately. The graphs show that clearly because it has the highest accuracy and the lowest RMSE. Hence, I have decided to keep Leaky ReLU and Adam for the model because they seem to work the best.

Graph 1: Activation function and Optimizer Exploration



3.3 Loss function description

To evaluate the performance of the model, I employ specific mathematical functions known as loss functions. These functions quantify the disparity between the model's predictions and the actual values, providing crucial feedback for refining the model's parameters. For the regression task of predicting house prices, I utilize the Mean Squared Error (MSE) as the loss function. The Mean Squared Error (MSE) measures the average squared difference between the predicted and actual values in a regression task. It is calculated by taking the square of the difference between each predicted value (ŷi) and its corresponding actual value (yi), summing these squared differences across all data points, and then dividing by the total number of data points (n): MSE = (1/n) * Σ(ŷi - yi)^2.

On the other hand, for the classification task, the Cross-Entropy loss function is utilized. The Cross-Entropy Loss, often used in classification tasks, quantifies the difference between the predicted probability distribution over classes and the actual distribution of class labels. It's calculated by taking the negative logarithm of the predicted probability assigned to the true class label. The formula for multi-class cross-entropy loss is = - (1/N) * Σ Σ y_ic * log(p_ic). In essence, the cross-entropy loss penalizes models more heavily for confidently incorrect predictions, thereby encouraging the model to output higher probabilities for the correct class. A lower cross-entropy loss indicates better alignment between predicted and actual class probabilities.

To combine both the Mean Squared Error (MSE) loss for the regression task and the Cross-Entropy loss for the classification task into a single loss function, I used a weighted sum approach. The individual losses were weighted differently to balance their contributions to the overall loss. For the regression task, I applied a very small weight to the MSE loss, because the sales prices of houses are very high, hence I wanted a smaller impact on the overall loss function. The combined loss function was formulated as follows: Combined Loss = (weight_regression * MSE_loss) + (weight_classification * CrossEntropy_loss). Here, weight_regression and weight_classification represent the weights assigned to the MSE loss and Cross-Entropy loss, respectively. By adjusting these weights, I could control the relative influence of each task on the model's training process. I tried running the model with different weights and did not see much of a difference in final results, hence I did not spend too much time deciding on optimal weights.

Section 4. Implementation and Training

4.1 Configuration of PyTorch Lightning

In this section, I configured PyTorch Lightning for the multi-task model training. PyTorch Lightning is a lightweight wrapper around PyTorch that simplifies the training process by providing pre-defined training loops and handling low-level details such as gradient updates and device placement. Firstly, I defined the model architecture using the MultiTaskModel class, which I previously described in Section 3. The model consists of a shared feature extraction module followed by task-specific output heads for regression (predicting house prices) and classification (predicting house categories). Next, I set up logging and callbacks for monitoring the training progress and saving the best model checkpoints. I used the TensorBoardLogger to visualize training metrics and the ModelCheckpoint callback to save the model with the lowest training loss. Then, I converted the input data from pandas DataFrames to PyTorch tensors using torch.tensor. This step was necessary to ensure compatibility with PyTorch's tensor-based operations and data loaders.

After converting the data, I created training and testing datasets using TensorDataset and created corresponding data loaders using DataLoader. Data loaders enable efficient batch processing and shuffling during training.

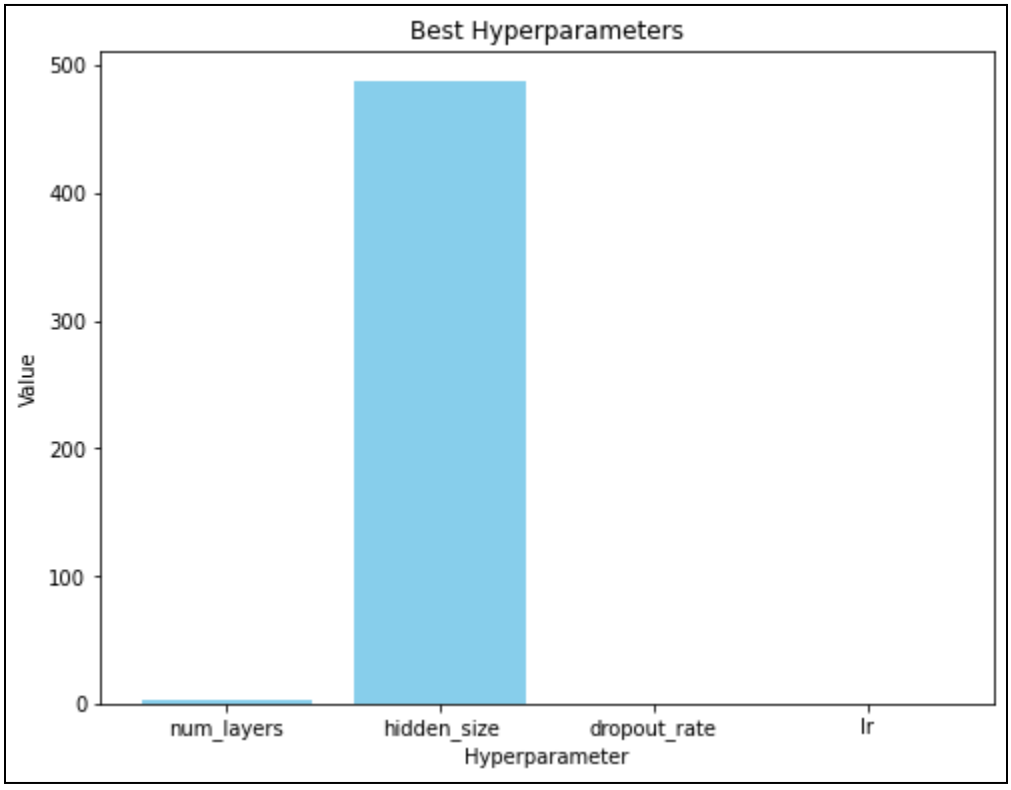
4.2 Techniques used for hyperparameter tuning

In this section, I used Optuna, a hyperparameter optimization framework, to search for the best combination of hyperparameters for the multi-task model. Optuna offers an efficient and flexible approach to hyperparameter tuning by using an adaptive search algorithm to explore the hyperparameter space and find the configuration that minimizes the loss function.

To integrate Optuna with PyTorch Lightning, I defined an Objective function named objective, which takes trial parameters as input and returns the loss to be minimized. Inside the objective function, I defined the search space for hyperparameters such as the number of layers, hidden size, dropout rate, and learning rate. I then set-up the multi-task model with the suggested hyperparameters and trained it using PyTorch Lightning. During the training process, the OptunaCallback class reports the training loss to Optuna at the end of each training epoch, allowing Optuna to adjust the search based on the reported loss values.

After running the Optuna optimization for a specified number of trials (in this case, 100), I obtained the best hyperparameters that minimized the loss. Graph 2 (seen below), displays the best values for each hyperparameter, including the number of layers, hidden size, dropout rate, and learning rate. The best hyperparameters found by Optuna were lr: 0.0099, hidden_size: 190, and dropout_rate: 0.1675. These hyperparameters represent the configuration that resulted in the lowest training loss during the hyperparameter tuning process.

Graph 2: Best Hyperparameter Values found with Optuna



Section 5. Results and Evaluation

5.1 Evaluation metrics used for both regression and classification tasks

For the regression task, I used several key metrics to evaluate the model's performance in predicting house prices. These metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) score. These metrics provide insights into the model's ability to accurately predict continuous numerical values, such as house prices. MSE, RMSE, and MAE quantify the differences between predicted and actual house prices, with lower values indicating better performance. Additionally, the R^2 score measures the proportion of variance explained by the model, offering a comprehensive assessment of its predictive power.

On the other hand, for the classification task, I used different metrics tailored to evaluating the model's performance in categorizing houses into distinct categories. The metrics were accuracy, precision, recall, and F1 score. Accuracy measures the overall correctness of the model's predictions, while precision quantifies the proportion of correctly predicted positive cases among all predicted positive cases. Recall, also known as sensitivity, calculates the proportion of correctly predicted positive cases among all actual positive cases. Lastly, the F1 score represents the harmonic mean of precision and recall, offering a balanced evaluation of the model's performance in classification tasks.

These metrics were chosen based on their relevance and effectiveness in evaluating the performance of regression and classification models. By calculating these metrics using the provided evaluation function, I will gain valuable insights into the strengths and weaknesses of the multi-task model across both regression and classification tasks, facilitating informed decision-making and model refinement.

5.2 Presentation of the training and validation results

The presented results show the performance of the multi-task model in predicting both house prices and house categories. In terms of regression metrics, the model shows an okay performance with a mean squared error (MSE) of approximately 1.66 billion, indicating the average squared difference between predicted and actual house prices. The root mean squared error (RMSE) of around 40,728.71 signifies the average deviation of predicted prices from actual prices, providing insight into the model's accuracy in predicting house prices. This is not that bad considering the average price for a home is around $180,921.195. Moreover, the mean absolute error (MAE) of approximately 25,866.94 shows the average absolute difference between predicted and actual prices. The model's ability to explain the variance in house prices is captured by the R-squared (R2) score of 0.7837, where higher values denote a better fit to the data. I am satisfied with the results as the R squared value shows that the independent variables explain the variability of the dependent variable pretty well.

On the classification side, the model achieves an accuracy of 27.4%, indicating the proportion of correctly classified house categories out of all predictions. The precision of approximately 7.79% measures the model's ability to correctly identify positive predictions from all predicted positive instances, while recall, also at 27.4%, quantifies the proportion of true positive predictions captured by the model. Furthermore, the F1-score, balancing precision and recall, stands at 12.13%, reflecting the harmonic mean of precision and recall. Overall, these results are not great for the classification task. I have tried playing around with the feature engineering and data preprocessing as well as the model architecture in order to increase this accuracy score, but with no luck. If I were to redo this project, I would spend more time investigating why my model is not accurately depicting the house category the majority of the time.

Section 6. Conclusion

6.1 Summary of findings and key insights

In conclusion, the analysis and modeling done in this project provide valuable insights into predicting house prices and categories. Through evaluation using various regression and classification metrics, I have gained a comprehensive understanding of the model's performance. The regression model shows a reasonably good fit to the data, as indicated by the relatively low mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) values. Additionally, the R^2 score suggests that a significant portion of the variance in house prices is explained by the model. However, the classification model's accuracy, precision, recall, and F1-score are relatively lower, indicating room for improvement in predicting house categories.

From a business perspective, these findings have important implications. A regression model can assist stakeholders, such as real estate agents and homeowners, in estimating property values more accurately, facilitating informed decision-making regarding pricing, buying, or selling properties. On the other hand, enhancing the classification model's performance can enable better categorization of houses based on various attributes, aiding in market segmentation and targeted marketing strategies. Overall, the insights derived from this project can empower stakeholders in the real estate industry to make better data-driven decisions, optimize resource allocation, and ultimately improve business outcomes.

Section 7. References

Anna Montoya, DataCanary. (2016). House Prices - Advanced Regression Techniques. Kaggle. https://kaggle.com/competitions/house-prices-advanced-regression-techniques