Multi-Task Learning for Predicting House Prices and Categories

Abdulrahman Aroworamimo

May 3, 2024

Contents

1 Introduction

In the dynamic and complex real estate market, accurate prediction of house prices and classification of house types are crucial for both market analysis and investment decisions. This project aims to build and evaluate a multi-task learning model that simultaneously predicts house prices, a regression task, and categorizes houses into predefined categories, a classification task. By leveraging multi-task learning models and utilizing advanced features of PyTorch Lightning, this project seeks to efficiently manage and optimize the learning process for these intertwined tasks.

2 Problem Statement

The challenge of this project is two-fold: to predict the sale price of houses and to classify them into categories based on age, building type, and renovation status. The goal is to develop a predictive model that can handle multiple output variables and exploit commonalities and differences between tasks to improve prediction accuracy and model efficiency.

3 Hypothesis

We hypothesize that the integration of house characteristics, such as building type and renovation history, into a unified model can significantly enhance the prediction performance over separate models handling each task independently. This hypothesis is based on the premise that certain features will have shared influence across both regression and classification tasks, leading to more robust and generalizable predictions.

4 Dataset

The dataset utilized in this analysis is the "House Prices - Advanced Regression Techniques" from Kaggle. In addition to the original features, this project introduces a new categorical variable, 'House Category', derived from the 'House Style', 'Bldg Type', 'Year Built', and 'Year Remod/Add' features. The categorization is as follows:

- Houses renovated or built within the last 20 years are categorized as 'Modern'.

- Houses renovated or built within the last 50 years are classified as 'Contemporary'.
- Houses renovated or built within the last 100 years fall under 'Vintage'.
- Older houses are labeled as 'Historic'.

Additionally, houses are categorized based on building type and style into 'Family Home', 'Townhouse', and 'Multi-Family or Duplex', among others. The combination of age and type categories results in the comprehensive 'House Category' for each property. This enriched dataset provides a nuanced framework for our multi-task learning model to operate within.

5 Data Preprocessing

The initial steps in data preprocessing involved a robust exploration of the dataset's characteristics and cleaning. Using `pandas_profiling.ProfileReport`, we conducted an exploratory data analysis that highlighted critical insights, particularly regarding the handling of null values as informed by the data dictionary.

5.1 Handling Missing Data

Significant portions of the dataset contained null values, which, rather than indicating missing data, provided meaningful information about the absence of certain features, such as pools or garages. These were handled as follows:

- Features indicating the presence of an attribute (e.g., `PoolQC`, `GarageType`) were filled with 'None' or similar indicators of non-existence.
- Numeric features like `GarageYrBlt` were filled with -1, and a binary indicator was created to denote the original presence of null values.

5.2 Feature Engineering and Reduction

Further preprocessing involved adjusting the granularity of categorical features by reducing sparse classes and managing highly correlated variables:

- Categorical variables were assessed for rare categories, which were then grouped into an 'Other' category to maintain statistical significance.
- Highly correlated numeric features (correlation greater than 0.7) were identified and redundant features were removed to avoid multicollinearity.

5.3 Encoding and Scaling

Categorical features were one-hot encoded to convert them into a format suitable for modeling:

- A `OneHotEncoder` was employed, set to drop the first category to avoid dummy variable trap.
- The dataset was then split into training, validation, and test sets to ensure the model could be evaluated on unseen data.
- Features were scaled using `MinMaxScaler` to ensure that the model was not biased by the scale of the data.

5.4 Dataset Preparation for Model Training

Finally, custom dataset classes were created for use with PyTorch, facilitating easy batch loading and data handling during model training:

- Datasets for training, validation, and testing were wrapped in `torch.utils.data.Dataset` objects.
- Data loaders were configured to shuffle training data and manage parallel loading using multiple workers.

This meticulous preprocessing and setup are crucial for the effective training of our multi-task learning model, ensuring that it learns to predict both house prices and categories accurately.

6 Architectures Tried

This section details the foundational architecture of the multi-task learning model developed using PyTorch Lightning, designed for simultaneous regression and classification tasks. Subsequent variations on this architecture are based on modifications to this foundational model.

6.1 Base Multi-Task Learning Model

The architecture employs a shared structure to extract features useful for both predicting house prices and categorizing house types, followed by task-specific outputs.

6.1.1 Shared Feature Layers

The model incorporates:

- A linear layer that expands the input features to 64 dimensions, followed by batch normalization and ReLU activation.
- A subsequent linear layer that reduces the dimensions to 32, also followed by batch normalization and ReLU activation.

These layers are designed to capture and transform the input data into a representation that supports both task-specific objectives.

6.1.2 Task-Specific Heads

The network bifurcates into two separate heads after the shared layers:

- **Price Head:** Comprises a linear reduction to 16 features, a ReLU activation, and a final linear output layer for price prediction (regression).
- **Category Head:** Mirrors the Price Head's structure but concludes with a linear layer designed for multi-class output (classification).

6.1.3 Loss Functions and Metrics

- The regression task utilizes the Mean Squared Error Loss (MSE).
- The classification task employs the Cross-Entropy Loss.
- Performance metrics include R2 Score for regression and accuracy, precision, recall, and F1 score for classification.

Notably, in this initial model configuration, losses from both tasks are not normalized or weighted differently, which may result in one task dominating the training process depending on the scale of error contributions from each task. This can potentially lead to suboptimal learning where one task's performance is enhanced at the expense of the other.

6.1.4 Optimization and Training

The model is optimized using an Adam optimizer with a learning rate of 0.001 and a StepLR scheduler that reduces this rate after each epoch. This setup aids in adjusting learning rates to balance learning speeds between tasks.

6.2 Weight Initialization

Weights in linear layers are initialized using the Kaiming normalization technique, which is suitable for layers followed by ReLU activations. Biases are initialized to zero. This method helps in maintaining a balanced variance in activations throughout the network, promoting stable and efficient learning dynamics.

6.3 Implications of Non-Normalized Losses

The absence of loss normalization in the initial architecture setup can lead to significant challenges in training convergence and performance. If the scale of losses between tasks differs greatly, the task with the higher scale may overshadow the learning signals from the other task, thereby skewing the model's ability to generalize effectively across tasks. Future architectural variations will explore methods to normalize or balance these losses to ensure equitable learning across both tasks.

6.4 Architectural Variations

Further architectures build upon this foundational model by introducing adjustments in layer configurations, activation functions, and loss normalization techniques to explore and enhance the model's performance and robustness in handling multi-task learning scenarios.

6.5 Architecture with Task Uncertainty Parameters

Building upon the base multi-task learning model, the second architectural variant incorporates task uncertainty parameters to address the challenge of non-normalized losses, which could potentially lead to the dominance of one task over another in terms of influence on the model's learning process.

6.5.1 Introduction of Uncertainty Parameters

The model architecture remains largely consistent with the shared layers and task-specific heads as defined in the base model. The significant enhancement involves the introduction of two parameters:

- **log_sigma_squared_price**: Represents the logarithm of the variance associated with the price prediction task.
- **log_sigma_squared_category**: Represents the logarithm of the variance associated with the house category classification task.

These parameters are integrated into the loss calculation to dynamically adjust the contribution of each task's loss to the overall training process based on the estimated uncertainty of each task.

6.5.2 Modified Loss Calculation

The loss functions for both tasks are adjusted to include a weighting mechanism governed by the uncertainty parameters:

- The regression loss (price prediction) and the classification loss (house category) are scaled inversely by their respective exponential uncertainty estimates.
- This scaling aims to balance the model's focus, reducing the weight of the loss for tasks where higher uncertainty is estimated, thereby mitigating the risk of one task overshadowing another.

6.5.3 Implications of This Approach

By dynamically adjusting the loss contributions from each task, the model is better positioned to:

- Achieve balanced learning between tasks.
- Improve convergence behavior by preventing the larger scale of one task's errors from dominating the gradient updates.
- Enhance the overall performance and generalization capability of the model across both tasks.

6.5.4 Training and Validation

The training and validation steps incorporate these modifications, with metrics and optimization procedures remaining consistent with the base model setup. The introduction of uncertainty parameters requires careful tuning and monitoring during training to ensure optimal balance and performance.

This variant demonstrates an advanced approach to tackling the challenges associated with multi-task learning, specifically in the context of loss normalization and task balancing. Future iterations and experiments will continue to refine this approach and explore additional modifications to enhance model robustness and accuracy.

6.6 Architecture with Elastic Net Regularization

The third variant of the multi-task learning model integrates Elastic Net regularization into the loss computation to manage the balance between L1 and L2 penalties, alongside employing task uncertainty parameters. This approach is intended to address both overfitting and the proper calibration of loss contributions from different tasks.

6.6.1 Elastic Net Integration

Elastic Net regularization is applied to both the price and category prediction tasks by combining L1 and L2 penalties, which are computed as follows:

- **Elastic Net Penalty:** Calculated as a weighted sum of the L1 and L2 norms of the model parameters, controlled by the hyperparameters α and $l1\_ratio$. This regularization aims to leverage the benefits of both Lasso (L1) and Ridge (L2) regularization techniques.

6.6.2 Loss Function Enhancement

The loss functions for both tasks include the uncertainty parameters and the elastic net penalty:

- The total loss is computed by adding the weighted losses of the price and category predictions with the elastic net penalty. This configuration ensures that the model does not overly focus on any single task or feature subset, promoting a balanced learning process.

6.6.3 Optimizer Variation

To further enhance model performance, two different optimizers were tested:

- **Adam Optimizer:** Initially, the model was trained using the Adam optimizer, known for its effective handling of sparse gradients.
- **AdamW Optimizer:** Subsequently, AdamW was employed, which modifies the way weight decay is applied and is believed to be more suitable for models with regularization terms.

6.6.4 Model Evaluation

The model's performance was rigorously evaluated across multiple metrics, including R2 Score for regression and accuracy, precision, recall, and F1 score for classification. The impact of different optimizers on these metrics was observed, providing insights into the optimizer's role in effective regularization.

6.6.5 Training and Validation Process

Throughout the training and validation phases, continuous monitoring and logging of various metrics were conducted to ensure that the model adjustments effectively addressed previous limitations and enhanced overall predictive performance.

This architectural variant demonstrates a sophisticated approach to multi-task learning, where the integration of elastic net regularization and the exploration of different optimization strategies are pivotal for advancing model robustness and accuracy.