Multi-Task Neural Networks for Predicting House Prices and Categories

Ethan Pirso

April 28, 2024

Introduction

The Multi-Task House Prediction project aims to concurrently predict house prices and classify houses into specific categories using a multi-task learning (MTL) framework. This project employs a sophisticated feed-forward neural network that utilizes a shared bottom model to process common features between the tasks, while employing task-specific top layers tailored to address the distinct demands of regression and classification tasks effectively.

This approach leverages the inherent efficiencies of MTL, where shared learning across related tasks can lead to a deeper and more nuanced understanding of data patterns. By training the model to perform both tasks simultaneously, it learns to generalize better, reducing the risk of overfitting to the idiosyncrasies of a single task dataset. This is particularly advantageous in complex real estate markets where variables influencing prices and categorical classifications—such s architectural styles or building types—are often interrelated.

The integration of a shared architecture for both price prediction and house categorization allows the model to extract and utilize latent representations that are beneficial for both tasks. This not only enhances the accuracy of predictions and classifications but also improves the computational efficiency of the model. The capability of MTL to exploit commonalities while simultaneously addressing differences across tasks positions it as an ideal choice for tackling the multifaceted challenges present in predicting diverse aspects of real estate properties. This project not only advances the field of real estate analytics but also demonstrates the broader applicability and robustness of multi-task learning models.

Model Architecture

The architecture of the Multi-Task Model in the house prediction project is meticulously designed to efficiently handle both regression and classification tasks, optimizing the learning process and performance across these different objectives. Below is a detailed breakdown of the model components:

Shared Layers

- **Structure**: The model starts with a sequence of linear layers that expand and then compress the feature dimensions, specifically from 63 to 256. This configuration allows the network to initially broaden its understanding of the input features and then refine this information into a more distilled form that's useful for both tasks.
- **Activation Functions**: A combination of ReLU and GELU activation functions is utilized within these layers. ReLU is employed initially to introduce non-linearity while maintaining

gradient flow, which is crucial for deep networks. GELU, known for capturing more complex patterns, is used deeper in the network to enhance the model's capability to learn from the data intricacies.

- **Normalization and Dropout**: Batch normalization is applied to stabilize learning and improve convergence speeds by normalizing layer inputs. Dropout follows each activation to prevent overfitting, ensuring the model generalizes well on unseen data.

Attention Mechanism

- **Multihead Attention**: This component is crucial for focusing the model on important features without explicit programming. It processes the shared layer output to enhance feature representation by considering the relationships between different positions in the input sequence. This is particularly useful in learning complex dependencies in data.

Transformer Encoder

- **Enhanced Processing**: Following the attention mechanism, the Transformer Encoder, composed of multiple layers, further processes these attention-enhanced features. Each encoder layer uses multi-head self-attention and is capable of handling sequences of data, making it incredibly effective in understanding spatial hierarchies in inputs.
- **Configuration**: The encoder includes several layers of TransformerEncoderLayer, each configured with a dimensionality of 256 and eight heads in multi-head attention, ensuring robust feature extraction across different parts of the input.

Task-Specific Heads

- **Regression Head**: Tailored specifically for price prediction, this head consists of several linear layers, each followed by GELU activation and dropout. This structure progressively refines the features into a single output value, representing the predicted price.
    - The layers are designed to decrease in size gradually, concentrating the learned features into precise predictions.
- **Classification Head**: For categorizing houses, this head starts with layers that expand the feature set before reducing them, similar to the regression head but adjusted to handle multiple class outputs. It uses LeakyReLU to enhance gradient flow, especially beneficial where standard ReLU might fail (e.g., with slightly negative values).
    - Final layers use GELU and dropout before culminating in a layer that outputs probabilities across eight categories, matched to the house types.

This architecture not only supports the nuanced demands of predicting continuous variables (prices) and class labels (categories) but also optimizes the shared use of learned features, reducing computational redundancy. The use of distinct task-specific heads after shared processing ensures that task-relevant features are effectively utilized, enhancing both the accuracy of price predictions and the precision of house categorizations.

Data Preprocessing

In preparing the data for the Multi-Task House Prediction project, a structured approach was adopted to meticulously preprocess the dataset, ensuring optimal readiness for training with our neural network. This process involved several critical steps aimed at refining the data for better model performance and handling of categorical variables using advanced embedding techniques.

Handling Missing Data and Redundant Features

The preprocessing commenced with the removal of columns that potentially could detract from model accuracy:

- **Missing Values**: Columns with more than 25% missing data were removed, under the premise that excessive imputation could introduce bias or inaccuracies.
- **Low Variance Features**: Columns where over 90% of the data points had zero values were also dropped. Such features offer little variability and are unlikely to contribute meaningful information to the model.

Feature Engineering

The first critical step involved creating new features that would allow the model to better understand and categorize the properties. A pivotal feature engineered was 'House Category', derived from existing data attributes including 'House Style', 'Building Type', 'Year Built', and 'Year Remod/Add'. This categorization aimed to reflect significant property characteristics that impact both price and type. For instance, properties were categorized into 'New' or 'Old' based on the relative age determined by the years since construction or last major remodel. This differentiation helps the model discern patterns associated with newer versus older homes, which often influence both market value and buyer interest.

Categorical Processing

The dataset contained a mix of numerical and categorical variables, requiring careful handling to ensure they were appropriately encoded:

- **Categorical Encoding**: Traditional methods like Ordinal Encoding were initially used to transform categorical strings into machine-readable integers. However, this approach was further enhanced by leveraging the capabilities of OpenAI's text embeddings.
- **OpenAI Text Embeddings**: To capture the nuanced relationships within categorical data more effectively, text embeddings were employed. By concatenating all categorical fields into a single string per record, embeddings were generated using OpenAI's API, which provides a dense representation of the input text. This method allows the model to understand deeper contextual relationships between categories compared to traditional encoding methods.

Normalization and Data Integration

- **Numerical Normalization**: Numerical columns were normalized using Min-Max scaling to ensure that all values were on a similar scale, thus preventing any single feature from dominating the model's learning process due to its scale.
- **Embedding Integration**: The embeddings obtained from the OpenAI API were scaled using Min-Max scaling to maintain consistency with the other numerical fields. These embeddings were then integrated back into the main dataset, replacing the original categorical columns.

Final Data Preparation

- **Imputation**: Missing values in both numerical and newly encoded categorical columns were imputed using the KNN Imputer, which infers the missing entries using the k-nearest neighbors.
- **Dataset Splitting**: If the dataset contained target variables (for the training set), these were separated from the features, ensuring the model could be trained to predict without bias.

This preprocessing pipeline not only ensured that the data fed into the model was clean and well-structured but also innovatively used advanced techniques like text embeddings to enhance the model's ability to interpret and learn from categorical data. These steps collectively set the stage for a robust training process, aiming to develop a model capable of effectively performing both regression and classification tasks in the housing market domain.

Training the Model

The training process for the Multi-Task House Prediction model was meticulously designed to optimize performance across both regression and classification tasks, leveraging the robust

capabilities of PyTorch Lightning. This process integrated advanced optimization strategies, tailored loss functions, and systematic hyperparameter tuning to enhance model accuracy and efficiency.

Optimization Strategy

To address the unique demands of the model's different components, a multi-optimizer strategy was implemented:

- **Shared Layers Optimizer**: An Adam optimizer with a learning rate of 0.005 was used for the shared layers, aiming to efficiently propagate updates through the foundational parts of the network that influence both task outcomes.
- **Task-Specific Optimizers**: Separate optimizers with tailored learning rates were employed for the regression head (0.0035) and the classification head (0.01). This approach allowed each task-specific component of the model to update its weights at a pace appropriate to its distinct learning dynamics, enhancing overall training effectiveness and addressing the different scales of data features and labels.

Loss Functions

The model used a combination of loss functions to handle the diverse nature of its tasks:

- **Regression Loss**: The Mean Squared Error Loss (MSE) was used for the regression task, providing a robust measure of model performance by penalizing the squared deviations of predicted house prices from their actual values.
- **Classification Loss**: Cross-Entropy Loss, weighted by class frequencies, addressed the class imbalance, ensuring that the model does not bias towards more frequent categories.
- **Loss Combination**: A scaling factor was introduced to balance the contributions of the regression and classification losses to the overall training loss, ensuring that neither task dominates the training process detrimentally.

Hyperparameter Tuning

Although fully systematic hyperparameter tuning using tools like Optuna was planned as a future enhancement, initial stages involved manual tuning and testing of parameters such as learning rates and dropout rates. The intended use of Optuna aims to automate this process, exploring a broad space of hyperparameters to identify the optimal configurations that minimize validation loss. This approach would potentially adjust parameters like the number of encoder layers, the dropout rate, and the number of attention heads, further refining the model's performance.

Training Execution

During training, each epoch involved calculating the root mean square error (RMSE) for regression and accuracy for classification, with these metrics logged for monitoring progress and performance tweaks. The model trained for 35 epochs, allowing sufficient iterations to converge on optimal solutions without overfitting.

This structured training approach, combined with strategic optimizer use, customized loss handling, and planned hyperparameter optimization, underpins the model's capability to effectively learn and predict diverse outcomes from complex housing market data.

Evaluation and Results

The Multi-Task House Prediction model, evaluated using RMSE for regression and accuracy for classification, achieved an RMSE of 60,196.13 and an accuracy of 83.90%. These results reflect the model's strong predictive ability, particularly in classifying house categories effectively. Validation was conducted using a hold-out strategy, affirming the model's capacity to generalize well to new, unseen data. Although there was no direct comparative analysis with baseline single-task models, it is anticipated that while there might be a slight decrease in computational efficiency due to the complexity of multi-task learning, the overall predictive power and insight into interdependent features are enhanced. Future efforts could focus on further optimizing RMSE through more sophisticated feature engineering and hyperparameter tuning, potentially exceeding the performance of traditional single-task approaches.

Conclusion and Future Work

The Multi-Task House Prediction project successfully developed a model that concurrently predicts house prices and categorizes properties using a multi-task learning framework. Key outcomes include achieving an RMSE of 60,196.13 for the price prediction task and an accuracy of 83.90% for the house categorization task. The model effectively leveraged a shared architecture to enhance learning efficiency and utilized task-specific optimizations to address the unique challenges of each prediction task.

Despite its successes, the project faced several limitations. The complexity of multi-task learning introduced challenges in balancing the training for both tasks without compromising the performance of one for the other. Additionally, the model's reliance on large volumes of high-quality data means that its performance is heavily contingent on the availability and variability of the input data, which can limit its applicability in areas with less data availability.

Future research could explore several avenues to enhance the model's performance and applicability. Incorporating more diverse datasets could help in improving the model's robustness and generalizability across different markets. Further development could also involve integrating more advanced machine learning techniques such as deep learning and reinforcement learning to refine predictions and handle more complex patterns. Experimenting with different architectural tweaks and additional features, such as more granular socio-economic indicators, could also potentially increase the accuracy and reliability of the predictions. Moreover, expanding the framework to include more tasks, such as estimating renovation costs or predicting future market trends, could greatly increase the utility of the model for real estate stakeholders.

Appendices

Category Number	Category Description
0	Multi-Level-Single-Family-New
1	Single-Level-Single-Family-New
2	Multi-Level-Single-Family-Old
3	Single-Level-Single-Family-Old
4	Single-Level-Townhouse-New
5	Multi-Level-Townhouse-New
6	Multi-Level-Townhouse-Old
7	Single-Level-Townhouse-Old

Figure 1: Housing categories.

```
  | Name                | Type               | Params
------------------------------------------------------------
0 | shared_layers       | Sequential         | 107 K
1 | attention           | MultiheadAttention | 263 K
2 | attention_linear    | Linear             | 65.8 K
3 | transformer_encoder | TransformerEncoder | 4.7 M
4 | regression_head     | Sequential         | 1.6 M
5 | classification_head | Sequential         | 1.3 M
------------------------------------------------------------
8.1 M     Trainable params
0         Non-trainable params
8.1 M     Total params
32.303    Total estimated model params size (MB)
```

Figure 2: Model architecture and parameters.

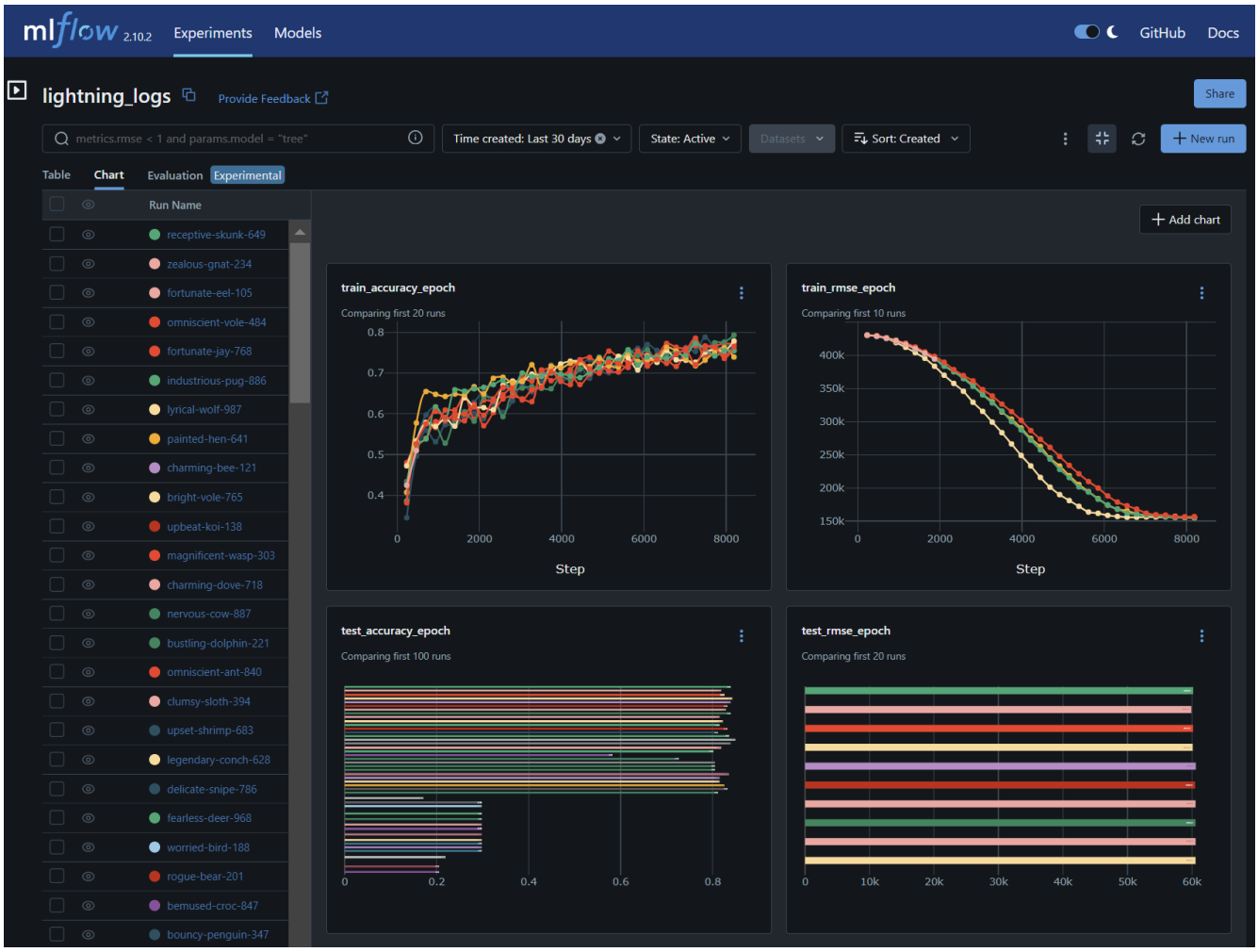

Figure 3: Training metrics and curves, logged in MLFlow.

Test metric	DataLoader 0
test_accuracy_epoch	0.8390411138534546
test_rmse_epoch	60196.12890625

Figure 4: Test metrics.