Comparative Analysis of Machine Learning Algorithms for House Price Prediction

Janvi Nandwani
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC, USA
jnandwan@charlotte.edu

***Abstract*—In this paper, I compare classical and advanced machine learning algorithms for predicting residential property prices using the Ames Housing dataset. I implemented three classical algorithms (Linear Regression, Polynomial Regression, and Neural Networks) and two literature-based methods (XGBoost and Stacking Ensemble) from recent research. My experiments show that XGBoost achieves the best performance with a test RMSE of $23,967.34, MSE of $574,433,344, MAE of $15,254.46, and R^2 of 0.9251, though I was surprised to find that simple linear regression with Ridge regularization came close with R^2 of 0.8712, RMSE of $23,932.69, and MAE of $16,016.72. Feature importance analysis reveals that garage capacity, overall quality, and living area are the strongest predictors. These findings suggest that while complex models like XGBoost excel, properly regularized linear models remain competitive for this task.**

***Index Terms*—machine learning, house price prediction, regression, XGBoost, ensemble methods, neural networks**

I. Introduction

Accurate house price predictions matter a lot in real estate. When buyers make one of the biggest purchases of their lives, or when banks decide on mortgage approvals, having reliable price estimates is essential. The traditional approach relies on human appraisers who physically inspect properties and make judgments based on experience. While this works, it's slow and different appraisers often come up with different values for the same house. Machine learning offers a way to make this process faster and more consistent by learning patterns from thousands of past sales [1].

A. Problem Statement

This project compares different machine learning approaches for predicting house prices. My main goals are to: (1) implement three classical algorithms that I learned in class, linear regression, polynomial regression, and neural networks, (2) find and reproduce two recent methods from research papers published after 2020, and (3) figure out which approaches work best and why. By testing multiple algorithms on the same dataset, I can see the tradeoffs between accuracy, complexity, and interpretability.

B. Dataset Overview

I used the Ames Housing dataset from Kaggle's "House Prices: Advanced Regression Techniques" competition. The dataset contains 1,460 residential property sales in Ames, Iowa, with 79 explanatory variables describing various aspects of residential homes. Features include:

- Physical attributes: Lot size (8,450 sq ft mean), living area (1,500 sq ft mean), number of rooms, garage capacity
- Quality ratings: Overall quality (1-10 scale), overall condition, kitchen quality, exterior quality
- Temporal features: Year built (1872-2010), year remodeled, year sold (2006-2010)
- Categorical variables: Neighborhood (25 categories), building type, roof style, foundation type

The target variable is sale price in US dollars, ranging from $34,900 to $755,000 with a median of $163,000. The dataset presents challenges including missing values (varies by feature), high dimensionality after encoding (260 features), and mixed feature types requiring careful preprocessing.

II. Related Work

Several recent papers have tackled housing price prediction with machine learning. Phan's 2024 paper [1] used XGBoost on this exact same Ames Housing dataset and showed it outperforms traditional methods like linear regression. What I found interesting about their approach was the focus on hyperparameter tuning and feature importance, they didn't just report accuracy numbers but actually explained which features matter most for pricing.

Truong et al. [2] took a different approach in 2020 by combining multiple models together. Their stacking ensemble used Random Forest, XGBoost, and LightGBM as base models, then trained a Ridge regression model on top to combine their predictions. The idea is that different models make different types of errors, so combining them can potentially give better results than any single model. However, they noted this comes with computational costs since you're training multiple models.

The classical methods, linear regression, polynomial regression, and neural networks, are still worth studying as baselines. Ridge regularization helps prevent overfitting when you have lots of features [3]. Neural networks can learn complex nonlinear patterns [4], though they require more data and careful tuning to work well. These fundamental approaches provide important reference points for evaluating more sophisticated techniques.

III. Methodology

A. Data Preprocessing and Feature Engineering

My preprocessing pipeline transforms the raw Ames Housing dataset into a format suitable for machine learning:

Missing Value Handling:

- Numerical features: Median imputation (e.g., LotFrontage, GarageArea)
- Categorical features: Filled with 'None' (e.g., NA in PoolQC indicates no pool)

Feature Encoding: I applied one-hot encoding to all 43 categorical variables, creating binary indicator features. Using `drop_first=True` reduced multicollinearity, resulting in 260 final features from 79 original variables.

Feature Scaling: Standardization ($\mu = 0, \sigma = 1$) was applied to all features using StandardScaler [5], fitted only on training data to prevent data leakage.

Data Splitting:

- Linear Regression: 80% train (1,168), 20% test (292)
- Polynomial/Neural Network: 64% train (934), 16% validation (234), 20% test (292)
- Paper methods: 80% train (1,168), 20% test (292)

Feature Selection (Polynomial only): To prevent dimensionality explosion with polynomial features, I selected the top 15 features by Pearson correlation with sale price: OverallQual (0.778), GrLivArea (0.675), GarageCars (0.638), GarageArea (0.617), TotalBsmtSF (0.593), among others.

B. Classical ML Algorithms

1) Linear Regression: I implemented linear regression with Ridge regularization ($\alpha = 10$) using both closed-form solution and gradient descent. The closed-form solution solves:

$$\theta = (X^T X + \alpha I)^{-1} X^T y \tag{1}$$

where θ are the learned weights, X is the feature matrix, y is the target vector, and α is the regularization parameter.

2) Polynomial Regression: To capture non-linear relationships, I expanded the top 15 most correlated features to polynomial degree 3. Ridge regularization ($\alpha = 10$) was applied to prevent overfitting. Hyperparameter tuning evaluated degrees [1, 2, 3] and α values [0.1, 1.0, 10.0, 100.0] using scikit-learn's Ridge implementation [5].

3) Neural Network: I built a multi-layer perceptron with ReLU activation and early stopping from scratch using NumPy [6]. Architecture search evaluated configurations: (64), (100), (128), (100, 50), (128, 64). The optimal architecture (100 hidden units) was trained using gradient descent with learning rate 0.01. Target normalization was applied for stable gradient descent.

C. Literature-Based Methods

1) XGBoost (Phan, 2024): **Paper Summary:** Phan's work "An Optimal House Price Prediction Algorithm: XGBoost" [1] compared multiple algorithms on the Ames Housing dataset and demonstrated XGBoost's superiority through extensive hyperparameter tuning. The paper emphasized feature importance analysis for interpretability.

Implementation: I reproduced their methodology using XGBoost [7] with GridSearchCV over 81 hyperparameter combinations (5-fold CV, 405 total fits). The search space included:

- `max_depth`: [3, 5, 7] - tree depth controlling model complexity
- `learning_rate`: [0.01, 0.05, 0.1] - step size for weight updates
- `n_estimators`: [100, 500, 1000] - number of boosting rounds
- `subsample`: [0.8] - fraction of samples for each tree
- `colsample_bytree`: [0.8] - fraction of features for each tree
- `min_child_weight`: [1, 3, 5] - minimum sum of instance weights in child

Best parameters: `max_depth=3`, `learning_rate=0.05`, `n_estimators=1000`, achieving CV RMSE of $27,449. I also replicated the feature importance analysis using XGBoost's built-in importance scores.

2) Stacking Ensemble (Truong et al., 2020): **Paper Summary:** Truong et al.'s "Housing Price Prediction via Improved Machine Learning Techniques" [2] proposed a stacking ensemble combining Random Forest, XGBoost, and LightGBM. The two-level approach uses diverse base learners (level 0) and a Ridge meta-learner (level 1) to combine predictions.

Implementation: I implemented the stacking framework using scikit-learn's StackingRegressor [5] with LightGBM [8]:

- **Base learners (Level 0):**
  - Random Forest: 200 estimators, `max_depth=15`, `min_samples_split=5`
  - XGBoost: 200 estimators, `max_depth=5`, `learning_rate=0.1`
  - LightGBM: 200 estimators, `max_depth=5`, `num_leaves=31`
- **Meta-learner (Level 1):** Ridge regression with $\alpha = 1.0$
- **Meta-feature generation:** 5-fold cross-validation to create out-of-fold predictions

The stacking approach trains base models independently, then uses their predictions as features for the meta-learner. This enables the meta-learner to learn optimal weights for combining diverse models.

IV. Results and Evaluation

A. Performance Comparison

Table I summarizes the performance of all five methods across multiple metrics. XGBoost achieved the best test performance with MSE of $574,433,344, RMSE of $23,967, MAE of $15,254, and R^2 of 0.925, demonstrating superior predictive accuracy. Surprisingly, linear regression performed nearly as well as the second-best model (MSE $572,774K, RMSE $23,933, MAE $16,017, R^2 0.871), while the classical neural network achieved competitive results (MSE $1,034M, RMSE $32,150, MAE $19,178, R^2 0.865).

TABLE I
MODEL PERFORMANCE COMPARISON

Model	Test MSE	Test RMSE	Test MAE	Test R^2
Linear Reg.	\$572.8M	\$23,933	\$16,017	0.871
Polynomial Reg.	\$1,257.1M	\$35,456	\$22,529	0.836
Neural Network	\$1,033.6M	\$32,150	\$19,178	0.865
XGBoost (Paper)	\$574.4M	\$23,967	\$15,254	0.925
Stacking (Paper)	\$783.7M	\$27,995	\$16,339	0.898

B. Evaluation Metrics Interpretation

Mean Squared Error (MSE): Measures average squared prediction error. Lower values indicate better performance. XGBoost's MSE of \$574.4M means squared errors average this value.

Root Mean Squared Error (RMSE): Square root of MSE, in same units as target. XGBoost's RMSE of \$23,967 means predictions are typically within \$24k of actual prices.

Mean Absolute Error (MAE): Average absolute prediction error. XGBoost's MAE of \$15,254 indicates the typical magnitude of prediction errors without squaring.

R^2 Score: Proportion of variance explained by the model (0-1 scale). XGBoost explains 92.5% of price variance, indicating excellent fit.

Train-Test Gap: Difference between training and test R^2 indicates overfitting. Linear regression shows minimal gap (0.037), while XGBoost shows larger gap (0.070) but maintains superior test performance.

C. Feature Importance Analysis

XGBoost's feature importance analysis reveals the most influential predictors of house prices. The top 15 features account for approximately 70% of total importance (Table II).

TABLE II
TOP 10 FEATURE IMPORTANCES FROM XGBOOST

Rank	Feature	Importance
1	GarageCars	0.1407
2	OverallQual	0.1162
3	ExterQual_TA	0.1023
4	GarageFinish_Unf	0.0984
5	FullBath	0.0428
6	MSZoning_RM	0.0374
7	GrLivArea	0.0329
8	KitchenQual_TA	0.0321
9	FireplaceQu_None	0.0227
10	Alley_None	0.0211

These results align with real estate intuition: garage capacity, overall quality, and living area are primary price drivers. The presence of quality-related categorical features (ExterQual_TA, KitchenQual_TA) demonstrates the importance of condition ratings in valuation.

D. Model-Specific Observations

Linear Regression: The closed-form solution outperformed gradient descent (RMSE \$23,933 vs \$26,199, MAE \$16,017 vs \$18,109), likely due to better convergence. Low overfitting gap (0.037) indicates good generalization despite model simplicity.

Polynomial Regression: Degree 3 with $\alpha = 10$ achieved best validation performance. However, test MSE (\$1,257M) and MAE (\$22,529) were higher than linear regression, suggesting insufficient regularization or feature selection for the expanded feature space.

Neural Network: Single hidden layer (100 units) outperformed deeper architectures, which showed signs of overfitting (validation RMSE 1.05 for 2-layer networks). Early stopping at 1000 epochs prevented overtraining. MAE of \$19,178 shows reasonable typical error magnitude.

XGBoost: Optimal hyperparameters (depth=3, lr=0.05, n=1000) achieved excellent performance with minimal overfitting (train MSE \$32.6M, test MSE \$574.4M). Feature importance analysis confirmed interpretability alongside accuracy. Low MAE (\$15,254) demonstrates consistently accurate predictions.

Stacking Ensemble: Despite combining three strong base learners, the ensemble (MSE \$783.7M, RMSE \$27,995, MAE \$16,339) didn't outperform XGBoost alone. This suggests the base models may not provide sufficiently diverse predictions for effective stacking, or that the meta-learner requires additional tuning.

V. DISCUSSION

A. Why Some Models Performed Better

XGBoost's Superiority: XGBoost achieved the best performance (R^2 = 0.925, MAE = \$15,254) due to several advantages: First, non-linear modeling captures complex interactions between features through tree-based splits. Second, it natively handles mixed types, processing categorical and numerical features without extensive preprocessing. Third, built-in L1/L2 penalties and tree-depth constraints prevent overfitting. Finally, gradient boosting sequentially corrects errors from previous trees, focusing on difficult predictions.

Linear Regression's Strong Performance: Despite its simplicity, linear regression achieved the second-best RMSE (\$23,933) and competitive MAE (\$16,017), nearly matching XGBoost. This success comes from Ridge regularization ($\alpha = 10$) effectively controlling overfitting in the 260-dimensional space. Also, sale prices follow an approximately log-normal distribution suited for linear models, and one-hot encoding captures categorical relationships linearly. The minimal train-test gap (0.037) indicates excellent generalization.

Neural Network's Moderate Success: The single-hidden-layer network (R^2 = 0.865, MAE = \$19,178) performed well but showed signs of overfitting (train R^2 = 0.950). Deeper architectures (100,50) and (128,64) drastically underperformed (validation RMSE > 1.0), likely due to insufficient training data (934 samples) for parameter estimation. Target normalization ($\mu = 0, \sigma = 1$) was crucial for stable gradient descent, and early stopping at 1000 epochs prevented further overtraining.

Polynomial Regression's Underperformance: Despite selecting top 15 features and using Ridge regularization, polynomial regression achieved the worst test MSE (\$1,257M) and

highest MAE ($22,529). Expanding 15 features to degree 3 creates 46 features, potentially introducing multicollinearity. Also, $\alpha = 10$ may be too weak for the expanded space, and limited training data (934 samples) relative to 46 parameters hampers stable estimation.

Stacking Ensemble Paradox: Surprisingly, stacking (R^2 = 0.898, MAE = $16,339) underperformed XGBoost alone (R^2 = 0.925, MAE = $15,254). This counterintuitive result suggests all three base learners (RF, XGBoost, LightGBM) are tree-based methods making similar errors, so there's a lack of diversity. Simple Ridge regression may insufficiently exploit base predictions, and combining three complex models increases variance without sufficient bias reduction.

B. Strengths and Limitations of Paper Methods

XGBoost (Phan, 2024) - Strengths:

- Comprehensive hyperparameter search (81 combinations) ensures near-optimal configuration
- Feature importance provides interpretable insights for stakeholders
- Reproducible methodology with clear parameter specifications
- Excellent performance-interpretability trade-off
- Low MAE indicates consistent prediction accuracy

XGBoost - Limitations:

- Computationally expensive: GridSearchCV required 15-20 minutes on modern hardware
- Hyperparameter sensitivity: sub-optimal choices significantly degrade performance
- Despite feature importance, individual prediction explanations remain opaque
- Requires careful tuning for each new dataset

Stacking Ensemble (Truong et al., 2020) - Strengths:

- Theoretically sound approach leveraging model diversity
- Combines complementary strengths of different algorithms
- Generalizes well when base models are truly diverse
- Provides robustness against individual model failures

Stacking - Limitations:

- Didn't outperform XGBoost alone in my implementation
- Highest computational cost: trains multiple models plus meta-learner
- Increased complexity with minimal accuracy gain over XGBoost
- Requires careful selection of diverse base learners
- Meta-learner choice (Ridge) may be suboptimal for this problem
- Higher MAE than XGBoost despite ensemble approach

C. Challenges During Reproduction

Hyperparameter Search Complexity: The XGBoost paper's extensive search space (81 combinations × 5 folds = 405 fits) required significant computation time. I maintained the search space to preserve fidelity to the original methodology despite the computational cost.

Stacking Implementation Details: The Truong et al. paper didn't specify exact hyperparameters for base learners. I made reasonable choices (200 estimators, depth 15 for RF) based on common practices, which may differ from the original implementation.

Feature Engineering Decisions: The papers provided limited detail on preprocessing. I used a standardized approach across all models, though paper-specific preprocessing might improve results.

Computational Resources: Training the stacking ensemble consumed significant memory (>8GB) due to storing out-of-fold predictions from 5-fold CV across three base models. This highlights practical deployment challenges for ensemble methods.

Validation Strategy: Different models used different splits (train/test vs train/val/test). While this reflects typical practices for each method, it complicates direct comparison. Future work should use identical splits across all models.

VI. Conclusion

A. Summary of Key Insights

This study comprehensively compared five machine learning approaches for residential property price prediction, yielding several important insights:

Algorithm Performance: XGBoost achieved state-of-the-art performance (MSE $574.4M, RMSE $23,967, MAE $15,254, R^2 = 0.925) through gradient boosting and extensive hyperparameter tuning. Surprisingly, classical linear regression with Ridge regularization performed nearly as well (MSE $572.8M, RMSE $23,933, MAE $16,017, R^2 = 0.871), showing that simpler models shouldn't be dismissed when properly regularized.

Model Complexity vs. Performance: Increased model complexity doesn't guarantee superior results. The stacking ensemble, despite combining three sophisticated models, underperformed XGBoost alone (R^2 = 0.898 vs. 0.925, MAE $16,339 vs. $15,254). Similarly, deeper neural networks (2 hidden layers) drastically underperformed single-layer architectures (validation RMSE 1.05 vs. 0.37), highlighting the importance of matching model complexity to dataset size.

Feature Importance: XGBoost's interpretability analysis revealed that garage capacity (14.1%), overall quality (11.6%), and exterior quality (10.2%) are the strongest price predictors. This provides actionable insights for real estate professionals and validates the model's learned patterns against domain knowledge.

Regularization Criticality: Across all models, regularization proved essential for generalization. Linear regression's Ridge penalty prevented overfitting in 260 dimensions, while polynomial regression's relatively weak regularization ($\alpha = 10$) contributed to its poor performance (MAE $22,529) despite feature expansion.

Practical Trade-offs: Model selection involves balancing accuracy, interpretability, and computational cost. XGBoost offers the best accuracy (MAE $15,254) but requires 15-20 minutes for hyperparameter tuning. Linear regression provides

90% of XGBoost's performance with <1 second training time, making it preferable for rapid prototyping or resource-constrained environments.

B. Future Work Suggestions

Several directions could extend and improve this research:

Advanced Feature Engineering: I used standard preprocessing, but domain-specific features could improve all models. Examples include: interaction terms between quality and size features, neighborhood-specific price trends, temporal features capturing market seasonality, and polynomial expansions of carefully selected continuous variables.

Temporal Validation: The random train-test split doesn't reflect real-world deployment where models predict future prices from historical data. Implementing time-based splitting (e.g., train on 2006-2008, test on 2009-2010) would better assess model robustness and practical utility.

Ensemble Refinement: The stacking ensemble's underperformance suggests opportunities for improvement: incorporating diverse base learners (e.g., linear models alongside tree-based), exploring alternative meta-learners (neural networks, gradient boosting), tuning base learner hyperparameters specifically for ensemble performance, and implementing weighted averaging based on validation performance.

Geographic Generalization: Testing models across different housing markets (urban vs. suburban, different states/countries) would assess transferability. Fine-tuning pretrained models on new regions could enable rapid deployment with minimal data.

Explainability Techniques: While XGBoost provides feature importance, individual prediction explanations remain limited. Implementing SHAP (SHapley Additive exPlanations) values could enable instance-level interpretability, crucial for high-stakes applications like mortgage approvals.

Uncertainty Quantification: Point predictions lack confidence intervals. Implementing conformal prediction or Bayesian approaches would provide prediction intervals (e.g., \$150k ± \$25k), enabling more informed decision-making.

In conclusion, this work shows that gradient boosting (XGBoost) achieves superior accuracy for housing price prediction, but classical methods remain valuable for interpretability and efficiency. The choice of algorithm should be guided by specific application requirements, balancing accuracy, interpretability, computational resources, and deployment constraints.

VII. Implementation Code

All code for this project is available in the GitHub repository:

`https://github.com/JanviN625/IntroMLCapstone`

The repository contains five Python scripts implementing the models:

- `linear_regression.py` - Ridge-regularized linear regression with closed-form and gradient descent solutions
- `polynomial_regression.py` - Polynomial features (degree 3) with Ridge regularization
- `neural_network.py` - Multilayer perceptron implementation from scratch
- `xgboost_paper.py` - XGBoost with hyperparameter tuning (Phan, 2024)
- `stacking_paper.py` - Stacking ensemble with RF, XGBoost, and LightGBM (Truong et al., 2020)

Each script includes data preprocessing, model training, evaluation, and visualization. Results are saved to the `results/` directory with corresponding CSV files and PNG plots. The repository also includes a detailed README with usage instructions and project structure.

Acknowledgment

This work was completed as part of the ITCS 5356 Introduction to Machine Learning capstone project at the University of North Carolina at Charlotte.

References

[1] T. D. Phan, "An Optimal House Price Prediction Algorithm: XGBoost," *Analytics*, vol. 3, no. 1, pp. 30-45, Jan. 2024.

[2] Q. Truong, M. Nguyen, H. Dang, and B. Mei, "Housing Price Prediction via Improved Machine Learning Techniques," *Procedia Computer Science*, vol. 174, pp. 433-442, 2020.

[3] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55-67, 1970.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[6] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357-362, 2020.

[7] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785-794.

[8] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 3146-3154.

[9] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

[10] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56-61.