Data Science & Machine Learning

Final Project Report

California Housing Prices Prediction

Prepared By:

Muhammad Osama

Abstract

This report explores the prediction of **median house values in California** using a dataset containing geographic, demographic, and economic features. Multiple regression algorithms—including Linear Regression, Lasso, Ridge, ElasticNet, Gradient Boosting, MLP, and SVR—were applied and evaluated. Gradient Boosting Regressor emerged as the best-performing model, achieving the lowest Mean Absolute Error (MAE: 31,709) and Mean Squared Error (MSE: 2.28×10^9). The report details data preprocessing, model selection, and performance analysis, providing insights into algorithmic effectiveness and recommendations for future improvements.

Introduction

1) Dataset Selection:

The California Housing Prices dataset was selected for its relevance to real-world housing markets and its inclusion of diverse features (e.g., location, income, room counts). Sourced from the 1990 U.S. Census, it contains 20,640 entries with 10 attributes, making it suitable for regression analysis. This data serves as an excellent introduction to implementing machine learning algorithms because it requires rudimentary data cleaning, has an easily understandable list of variables. The data contains information from the 1990 California census, although it may not help us with predicting current housing prices, it does provide an accessible introductory dataset for teaching people about the basics of machine learning.

2) Features of Data:

The dataset has total 10 features and the target variable is **medianHouseValue**

- **longitude:** A measure of how far west a house is; a higher value is farther west
- **latitude:** A measure of how far north a house is; a higher value is farther north
- **housingMedianAge:** Median age of a house within a block; a lower number is a newer building
- **totalRooms:** Total number of rooms within a block
- **totalBedrooms:** Total number of bedrooms within a block
- **population:** Total number of people residing within a block
- **households:** Total number of households, a group of people residing within a home unit, for a block
- **medianIncome:** Median income for households within a block of houses (measured in tens of thousands of US Dollars)
- **medianHouseValue:** Median house value for households within a block (measured in US Dollars)
- **oceanProximity:** Location of the house w.r.t ocean/sea

3) Correlation with Target Variable

Correlation refers to a statistical relationship between two or more variables. It tells you how strongly and in what direction two variables are related to each other.

Positive Correlation: When one variable increases, the other also increases.

Negative Correlation: When one variable increases, the other decreases.

No Correlation: No apparent relationship between the variables.

Feature	Correlation with Target:
median_house_value	1
median_income	0.688075
ocean_proximity_<1H OCEAN	0.256617
ocean_proximity_NEAR BAY	0.160284
ocean_proximity_NEAR OCEAN	0.141862
total_rooms	0.134153
housing_median_age	0.105623
households	0.065843
total_bedrooms	0.049686
ocean_proximity_ISLAND	0.023416
population	-0.02465
longitude	-0.045967
latitude	-0.14416
ocean_proximity_INLAND	-0.484859

This tells us that median_income is **positively correlated** with median_house_value. And ocean_proximity_INLAND is **negatively correlated** with median_house_value.

Methodology and Model Deployment

1) Data Preprocessing

Using isna().sum() to check the total number of missing values in the data set in this dataset in total_bedrooms column we have 207 missing values.

Column Name	Missing values
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	270
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity	0

Missing Values: Removed 207 rows with missing total_bedrooms values to ensure data integrity.

Duplicates: Dropped duplicate rows to prevent overfitting.

Categorical Encoding: One-hot encoded ocean_proximity (a nominal categorical variable) into boolean columns (e.g., INLAND, NEAR BAY).

Train-Test Split: Data shuffled and split into 80% training and 20% testing sets. For Ridge/Lasso, an additional 10% validation split was used for hyperparameter tuning.

2) Feature Engineering

Correlation Analysis: median_income showed the strongest positive correlation with median_house_value (0.69), while latitude had a weak negative correlation (-0.14).

Hot Encoding: Categorical values converted to Boolean values using hot encoding.

3) Algorithms Applied

1. Linear Regression:

Linear Regression models the relationship between independent variables and a target by fitting a linear equation (e.g., $y=\beta0+\beta1x1+\cdots+\beta nxn$$y=\beta0+\beta1x1+\cdots+\beta nxn$).

Application & Results:

- Used as a baseline model with no regularization.
- Achieved **MAE: 50,413** and **MSE: 4.80×10^{9}**, the third-worst performance.
- **Why It Underperformed**: Housing prices involve non-linear relationships (e.g., geographic location interacting with income) that a linear model cannot capture

2. Lasso Regression (L1 Regularization)

Lasso adds an L1 penalty to the loss function, shrinking some coefficients to zero. This performs feature selection.

Application & Results:

- Best alpha **(100)** selected via validation (MAE: 51,257).
- **Weak Performance:**
    - High alpha aggressively penalized coefficients, potentially removing useful features (e.g., latitude or households).

- Correlation analysis showed most features had low but non-zero importance; eliminating them hurt accuracy.

3. Ridge Regression (L2 Regularization)

Ridge adds an L2 penalty to shrink coefficients uniformly without eliminating them.

Application & Results:

- Best alpha (**10**) balanced bias and variance (MAE: 51,232).
- **Slightly Better Than Lasso**:
  - Retained all features but reduced overfitting by moderating coefficient magnitudes.
  - Features like median_income (high correlation) benefited from controlled shrinkage.

4. Elastic Net Regression

Combines L1 and L2 penalties balancing feature selection and coefficient shrinkage.

Application & Results:

- Used α=0.1 and *l*1_*ratio*=0.5 (equal emphasis on L1/L2).
- **MAE: 50,452** – marginally worse than Linear Regression.
- **Limitation**: Suboptimal hyperparameters. A grid search for alpha and l1_ratio might improve performance.

5. Gradient Boosting Regressor (GBR)

An ensemble method that builds decision trees sequentially, where each tree corrects errors from the previous one. Uses gradient descent to minimize loss.

Application & Results:

- **Best Model** (MAE: 31,709, MSE: 2.28×10^{9}).
- **Why It Excelled**:
  - **Non-Linear Modeling**: Captured interactions (e.g., high median_income + coastal proximity = higher prices).

- **Hyperparameter Tuning**: GridSearchCV optimized n_estimators=300, learning_rate=0.1, and max_depth=5, balancing complexity and generalization.
    - **Error Correction**: Iterative tree-building refined predictions progressively.

6. MLPRegressor (Neural Network)

A feedforward neural network with hidden layers. Uses backpropagation to adjust weights and minimize prediction error.

Application & Results:

- Architecture: 2 hidden layers (10 and 5 neurons), 5,000 iterations.
- **MAE: 48,249** – better than linear models but worse than GBR.
- **Limitations**:
    - **Shallow Network**: Limited capacity to model complex patterns.
    - **Training Constraints**: Fixed architecture and iterations; no hyperparameter tuning

7. Support Vector Regression (SVR)

Finds a hyperplane that minimizes prediction error within a margin. Kernel trick maps data to higher dimensions for non-linear modeling.

Application & Results:

- Used a **linear kernel** with C=100C=100 (low regularization) and ϵ=0.1ϵ=0.1.
- **MAE: 49,232** – moderate performance.
- **Why It Underperformed**:
    - **Kernel Choice**: A linear kernel cannot capture non-linear relationships. The
    - **Hyperparameter Sensitivity**: High C prioritized minimizing training error, risking overfitting.

4) Hyperparameter Tuning

How GridSearchCV Optimized GBR

GridSearchCV optimized the Gradient Boosting Regressor (GBR) by systematically evaluating hyperparameter combinations through K-fold cross-validation. The dataset was split into 5 folds (K=5), where each iteration trained the model on 4 folds and validated on

the 5th fold, repeating this process for all folds to ensure robust performance assessment. For GBR, GridSearchCV tested 27 combinations of hyperparameters:

n_estimators (100, 200, 300): Number of sequential decision trees.

learning_rate (0.01, 0.1, 0.2): Step size for error correction.

max_depth (3, 4, 5): Complexity of individual trees.

For each combination, the algorithm calculated the average validation Mean Squared Error (MSE) across all folds. The best-performing configuration (n_estimators=300, learning_rate=0.1, max_depth=5) minimized validation MSE, ensuring the model balanced complexity (via deeper trees) and generalization (via controlled learning rate). By leveraging K-fold cross-validation, GridSearchCV mitigated overfitting risks and identified hyperparameters that generalized well to unseen data, ultimately achieving the lowest test MAE (31,709) and MSE (2.28×10^9) for GBR.

Hyperparameters SVM

1. kernel defines what type of data pattern we want to capture rbf is used to capture non-linear data, linear can capture linear pattern in data set.

2. C is the regularization parameter which controls the tradeoff between model complexity and error. Smaller C for simpler models which have low tolerance for error, while large value minimizes the training error but has the risk of overfitting.

3. epsilon is the margin for no error penalty for example if epsilon is 0.1 and our Actual value is 10 and Predicted value is 10.08 so we have a margin of +- 0.1 and no loss will be counted.

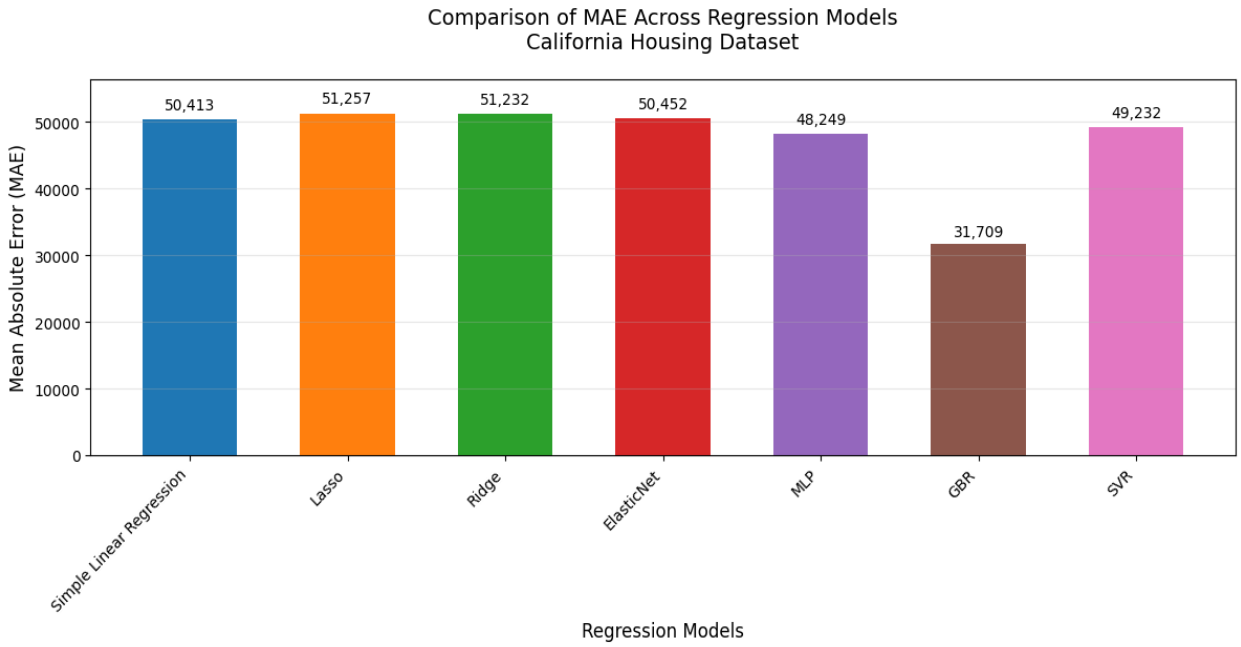
Hyperparameters MLP Regressor

1. hidden_layer_size this parameter defines the architecture of neural network, how many neurons and how many hidden layers, in this case there is 1 layer with 10 neurons and another hidden layer with 5 neurons. Increasing neurons and layers might help to capture complex patterns but if the data set is small more neurons and hidden layer might lead to overfitting.

2. max-iter defines the number of iterations the training algorithm will run while trying to find the best weights. The algo can run maximum number defined however if it converges means the change is negligible it will stop early before reaching maximum iteration. If max iter are too low the algorithm will not train properly, and the error will be huge.

Results and Analysis

1) Performance Metrics



Model	**MAE**	**MSE**
Gradient Boosting (GBR)	31,709	2.28×10^9
MLP	48,249	4.47×10^9
Linear Regression	50,413	4.80×10^9
ElasticNet	50,452	4.81×10^9
Ridge	51,232	5.01×10^9
Lasso	51,257	5.02×10^9
Support Vector Regression	49,232	5.02×10^9

The Gradient Boosting Regressor (GBR) achieved the best performance with a Mean Absolute Error (MAE) of 31,709 and Mean Squared Error (MSE) of 2.28×10^9, outperforming other models due to its ability to capture non-linear relationships (e.g., interactions between income and geographic location) and systematic hyperparameter tuning via GridSearchCV, which optimized tree depth, learning rate, and the number of estimators. Linear models (Linear, Lasso, Ridge, ElasticNet) and Support Vector Regression (SVR) underperformed due to their reliance on linear assumptions or suboptimal kernel choices, while the MLPRegressor's shallow architecture limited its capacity to model complex patterns. To further improve predictions, future efforts should prioritize hyperparameter tuning for

underperforming models (e.g., SVR with RBF kernel, deeper neural networks), advanced feature engineering (e.g., household density ratios, geographic binning), and integration of external data (e.g., crime rates, school quality).

2) Visualization

MAE/MSE Comparison: Bar plots above highlight GBR's superiority.

Actual vs. Predicted Plots: GBR predictions (see Notebook) clustered tightly around the ideal line, while linear models had wider dispersion.

3) Conclusion and Future Work:

While some steps ensured data readiness, more advanced feature engineering could further enhance model accuracy and interpretability. For instance:

- **Derived Ratios**: Features like bedrooms_per_room or population_per_household could better represent household density and property utility, directly influencing pricing.
- **Interaction Terms**: Combining median_income with ocean_proximity could model how income effects vary by location (e.g., high-income coastal vs. inland areas).

While GBR excelled despite limited feature engineering, simpler models like Linear Regression suffered due to their reliance on linear assumptions. Engineered features could bridge this gap by:

1. **Reducing Model Bias**: Providing richer context (e.g., household density ratios) to linear models.
2. **Enhancing Non-Linear Models**: Helping GBR or neural networks uncover subtler patterns (e.g., regional price clusters).

Future Improvements:

- **Advanced Engineering**: Create polynomial/interaction features (e.g., income × rooms_per_household) or temporal trends (e.g., inflation-adjusted prices).
- **Domain-Specific Features**: Integrate external data (e.g., school ratings, crime rates) to reflect real-world pricing factors.
- **Automated Feature Tools**: Use libraries like FeatureTools for automated feature generation from raw data.

By prioritizing these steps, future iterations could achieve even lower errors while providing actionable insights into the drivers of housing prices.