Lab Assignment Report

Assignment

CS503: Machine Learning

Name	Entry No.
Anshika	2021CSB1069

Instructor: Dr. Santosh Vipparthi

Teaching Assistant: Nishchala Thakur

Indian Institute of Technology Ropar
Punjab, India
March 30, 2025

Contents

float

1 | Optimization

1.1 | Key Analysis

Prove whether $L(\theta)$ is convex under linear models for both tasks
We begin by defining the linear model for regression:

$$f(x) = \theta^T x$$

where:

- $\theta \in \mathbb{R}^d$ is the parameter vector,
- $x \in \mathbb{R}^d$ is the input feature vector,
- $f(x) \in \mathbb{R}$ is the predicted output.

1.1.1 | Proof of Convexity of MSE loss function for regression

The MSE loss function for regression tasks is defined as:

$$L_r(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_r^{(i)} - \theta^T x^{(i)} \right)^2$$

where:

- n is the number of data points,
- $y_r^{(i)} \in \mathbb{R}$ is the true target for the i-th data point.

1. **Gradient of the MSE Loss Function:**
The gradient of the MSE loss function with respect to θ is:

$$\nabla_\theta L_r(\theta) = \frac{2}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} - y_r^{(i)} \right) x^{(i)}$$

2. **Hessian of the MSE Loss Function:**
The Hessian matrix, which is the second derivative of the loss function with respect to θ, is:

$$\nabla_\theta^2 L_r(\theta) = \frac{2}{n} \sum_{i=1}^{n} x^{(i)} \left(x^{(i)} \right)^T$$

Since the Hessian is a sum of outer products of the feature vectors $x^{(i)}$, it is a positive semidefinite matrix. Specifically, for any vector $z \in \mathbb{R}^d$, we have:

$$z^T \nabla_\theta^2 L_r(\theta) z = \frac{2}{n} \sum_{i=1}^{n} \left(z^T x^{(i)} \right)^2 \geq 0$$

Thus, the MSE loss function $L_r(\theta)$ is convex with respect to θ.

1.1.2 | Convexity of Cross-Entropy Loss for Classification

For binary classification, we define the logistic model as:

$$g(x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(x) \in (0, 1)$ is the predicted probability of the positive class.
The cross-entropy loss function is defined as:

$$L_c(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_c^{(i)} \log g(x^{(i)}) + (1 - y_c^{(i)}) \log \left(1 - g(x^{(i)}) \right) \right]$$

where $y_c^{(i)} \in \{0, 1\}$ is the true label for the i-th data point.
1. **Gradient of the Cross-Entropy Loss Function:**
The gradient of the cross-entropy loss function with respect to θ is:

$$\nabla_\theta L_c(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(g(x^{(i)}) - y_c^{(i)} \right) x^{(i)}$$

2. **Hessian of the Cross-Entropy Loss Function:**
The Hessian matrix is:

$$\nabla_\theta^2 L_c(\theta) = \frac{1}{n} \sum_{i=1}^{n} g(x^{(i)}) \left(1 - g(x^{(i)})\right) x^{(i)} \left(x^{(i)}\right)^T$$

Since $g(x^{(i)}) \left(1 - g(x^{(i)})\right) \geq 0$, the Hessian is positive semidefinite. For any vector $z \in \mathbb{R}^d$, we have:

$$z^T \nabla_\theta^2 L_c(\theta) z = \frac{1}{n} \sum_{i=1}^{n} g(x^{(i)}) \left(1 - g(x^{(i)})\right) \left(z^T x^{(i)}\right)^2 \geq 0$$

Therefore, the cross-entropy loss function $L_c(\theta)$ is convex with respect to θ.

1.1.3 | Convexity of the Joint Loss Function

The joint loss function that combines both the regression and classification losses is:

$$L(\theta) = \alpha L_r(\theta) + \beta L_c(\theta) + \lambda \|\theta\|_2^2$$

where α, β, and λ are non-negative constants. Since the sum of convex functions is convex, and the ℓ_2 norm regularization term $\|\theta\|_2^2$ is also convex, the joint loss function $L(\theta)$ is convex.

1.2 | Implications of Non-Convexity in Deep Neural Networks (DNNs)

Deep neural networks (DNNs) are non-convex due to their multiple layers and non-linear activations, which complicates optimization. Here's a brief breakdown:

1. **Non-Convexity**: A function is convex if a weighted average of two points is always less than or equal to the function's values at those points. DNNs, however, are non-convex, mainly due to non-linear activation functions like ReLU or sigmoid, leading to challenging optimization landscapes.
2. **Local Minima**: Non-convex functions have many local minima, which may trap gradient-based optimizers, making it hard to find a global minimum.
3. **Saddle Points**: Saddle points occur where the gradient is zero but aren't local minima. These points can slow optimization as the gradients around them vanish, making progress difficult.
4. **Gradient-Based Optimization**: Algorithms like gradient descent may converge to local minima or saddle points in non-convex functions, and performance depends heavily on initialization.
5. **Regularization and Generalization**: Non-convexity can lead to overfitting. Regularization techniques (e.g., ℓ_2 norm, dropout) are applied to improve generalization by adding a penalty to the loss function.
6. **Practical Strategies**: Techniques like good weight initialization, stochastic gradient descent (SGD), and learning rate schedules help DNNs perform well despite non-convexity challenges.

1.3 | Hessian Conditioning: Derivation of the Condition Number

Consider the Hessians $H_r = \nabla^2 L_r$ and $H_c = \nabla^2 L_c$ corresponding to two different loss functions, L_r and L_c, respectively. The joint Hessian is defined as the weighted sum of these two Hessians:

$$H = \alpha H_r + \beta H_c$$

where α and β are scalar coefficients that weight the contributions of H_r and H_c, respectively.

1. Definition of the Condition Number
The **condition number** $\kappa(H)$ of a matrix H (specifically for symmetric positive definite matrices like the Hessian) is defined as the ratio of the largest eigenvalue $\lambda_{\max}(H)$ to the smallest eigenvalue $\lambda_{\min}(H)$:

$$\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)}$$

The condition number provides a measure of the sensitivity of the system, i.e., how changes in the input affect changes in the output. A high condition number indicates that the matrix is ill-conditioned, meaning small changes in the input can cause large changes in the solution.

2. Joint Hessian Matrix

We now consider the joint Hessian $H = \alpha H_r + \beta H_c$. The eigenvalues of H are related to the eigenvalues of H_r and H_c. Let the eigenvalues of H_r and H_c be λ_r and λ_c, respectively. For the joint Hessian, the eigenvalues are a linear combination of those of H_r and H_c:

$$\lambda(H) = \alpha\lambda(H_r) + \beta\lambda(H_c)$$

Thus, the maximum eigenvalue of H, denoted $\lambda_{\max}(H)$, is given by:

$$\lambda_{\max}(H) = \alpha\lambda_{\max}(H_r) + \beta\lambda_{\max}(H_c)$$

Similarly, the minimum eigenvalue of H, denoted $\lambda_{\min}(H)$, is:

$$\lambda_{\min}(H) = \alpha\lambda_{\min}(H_r) + \beta\lambda_{\min}(H_c)$$

3. Condition Number of the Joint Hessian

Using the definition of the condition number, the condition number $\kappa(H)$ of the joint Hessian is given by the ratio of the maximum eigenvalue to the minimum eigenvalue:

$$\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)}$$

Substituting the expressions for $\lambda_{\max}(H)$ and $\lambda_{\min}(H)$ from the previous section, we get:

$$\kappa(H) = \frac{\alpha\lambda_{\max}(H_r) + \beta\lambda_{\max}(H_c)}{\alpha\lambda_{\min}(H_r) + \beta\lambda_{\min}(H_c)}$$

This equation gives the condition number of the joint Hessian $H = \alpha H_r + \beta H_c$, where the condition number is dependent on the individual condition numbers of H_r and H_c, as well as the weights α and β.

4. Special Cases

Case 1: $\alpha = 1, \beta = 0$
In this case, the joint Hessian simplifies to $H = H_r$, and the condition number reduces to:

$$\kappa(H) = \frac{\lambda_{\max}(H_r)}{\lambda_{\min}(H_r)}$$

Case 2: $\alpha = 0, \beta = 1$
Similarly, when $H = H_c$, the condition number is:

$$\kappa(H) = \frac{\lambda_{\max}(H_c)}{\lambda_{\min}(H_c)}$$

Case 3: $\alpha = \beta = 1$
If both Hessians are weighted equally, the condition number becomes:

$$\kappa(H) = \frac{\lambda_{\max}(H_r) + \lambda_{\max}(H_c)}{\lambda_{\min}(H_r) + \lambda_{\min}(H_c)}$$

The condition number of the joint Hessian $H = \alpha H_r + \beta H_c$ is derived based on the eigenvalues of the individual Hessians H_r and H_c. The condition number reflects the sensitivity of the optimization process, and it is crucial in understanding how well the joint Hessian is conditioned. The weights α and β play an important role in determining the overall condition number.

1.4 | Effect of Condition Number on Optimization Convergence

In optimization, the condition number of the Hessian matrix H, denoted as $\kappa(H)$, plays a crucial role in determining the convergence rate of second-order methods such as Newton's method. In this document, we will show how the condition number affects the convergence of optimization algorithms.

1.4.1 | Optimization and Hessian Matrix

Consider a convex objective function $L(\theta)$, where θ represents the model parameters. In second-order methods, such as Newton's method, the update rule is:

$$\theta_{k+1} = \theta_k - H^{-1}\nabla L(\theta_k)$$

Here, $H = \nabla^2 L(\theta_k)$ is the Hessian matrix at iteration k, and $\nabla L(\theta_k)$ is the gradient.

Condition Number
The condition number $\kappa(H)$ of the Hessian matrix is defined as:

$$\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)}$$

where $\lambda_{\max}(H)$ and $\lambda_{\min}(H)$ are the largest and smallest eigenvalues of H, respectively.

Effect on Convergence
The condition number $\kappa(H)$ affects the rate of convergence in the following ways:

- If $\kappa(H)$ is small (well-conditioned Hessian), the optimization will converge quickly, with updates moving uniformly in all directions.
- If $\kappa(H)$ is large (ill-conditioned Hessian), the optimization process will oscillate or make slow progress in certain directions, leading to slow convergence.

Convergence Rate
The number of iterations k required to achieve a certain accuracy ϵ in gradient-based methods is given by:

$$k = O\left(\kappa(H)\log\left(\frac{1}{\epsilon}\right)\right)$$

Thus, the convergence rate is directly proportional to the condition number. A larger condition number implies slower convergence, while a smaller condition number leads to faster convergence.

Scenario 1: Well-Conditioned Hessian
When $\lambda_{\max}(H) \approx \lambda_{\min}(H)$, the condition number $\kappa(H)$ is close to 1, leading to fast and uniform convergence in all directions.

Scenario 2: Ill-Conditioned Hessian
When $\lambda_{\max}(H) \gg \lambda_{\min}(H)$, the condition number is large, and the optimization process takes much longer to converge due to inefficient steps in certain directions.

The condition number $\kappa(H)$ of the Hessian matrix is a key factor in determining the convergence rate of optimization algorithms. A well-conditioned Hessian leads to faster convergence, while an ill-conditioned Hessian slows down the optimization process.

1.5 | L1 (sparse feature selection) vs. L2 (smooth feature weighting)

Regularization is a key technique used to prevent overfitting and improve generalization by adding a penalty term to the objective function. The two common types of regularization are:

- **L1 Regularization (Lasso)**: Encourages sparse feature selection.
- **L2 Regularization (Ridge)**: Encourages smooth feature weighting.

1.5.1 | L1 Regularization (Lasso)

L1 regularization adds the absolute value of the coefficients to the loss function:

$$L(\theta) = L_0(\theta) + \lambda\sum_{i=1}^{n}|\theta_i|$$

Effect of L1 Regularization

- **Sparsity**: L1 regularization tends to drive some coefficients θ_i to exactly zero, which leads to feature selection.
- **Feature Selection**: By forcing some coefficients to zero, L1 regularization simplifies the model by reducing the number of features used.

1.5.2 | L2 Regularization (Ridge)

L2 regularization adds the squared value of the coefficients to the loss function:

$$L(\theta) = L_0(\theta) + \lambda \sum_{i=1}^{n} \theta_i^2$$

Effect of L2 Regularization

- **Smooth Feature Weights**: L2 regularization does not force coefficients to zero but shrinks them toward zero.
- **Better Generalization**: L2 helps to smooth the model weights, preventing overfitting and reducing multicollinearity.

1.5.3 | Comparison: L1 vs. L2 Regularization

Property	L1 Regularization (Lasso)	L2 Regularization (Ridge)
Penalty Term	$\lambda \sum \lvert\theta_i\rvert$	$\lambda \sum \theta_i^2$
Feature Selection	Yes (produces sparse models)	No (produces dense models)
Coefficient Behavior	Drives some to zero	Shrinks, but not to zero
Optimization Path	Axis-aligned (sparse)	Spherical (smooth)

1.6 | Justify the choice of regularization strength λ using the bias-variance trade-off

- **Bias** refers to the error introduced by approximating the real-world problem with a simplified model. High bias models (underfitting) fail to capture the underlying patterns in the data.
- **Variance** refers to the model's sensitivity to small fluctuations in the training data. High variance models (overfitting) capture noise in the data, performing well on the training set but poorly on unseen data.

Regularization Strength λ The regularization term controls the model's complexity by penalizing large parameter values, thereby affecting both **bias** and **variance**. The role of λ in the bias-variance trade-off is as follows:

Small λ (Low Regularization)

- The penalty term is minimal, allowing the model to fit the training data closely.
- This leads to **low bias**, as the model can capture complex patterns and relationships in the data.
- However, it can result in **high variance**, as the model is more prone to overfitting, capturing noise and specific details in the training data.
- Model may perform well on the training data but poorly on new, unseen data due to overfitting.

Large λ (High Regularization)

- The penalty term is stronger, restricting the model from learning overly complex patterns.
- This leads to **high bias**, as the model may oversimplify the relationships, failing to capture important details.

- At the same time, it reduces **variance**, since the model is less sensitive to noise in the training data, making it more stable and generalizable.
- Model that might underfit the data, failing to capture the full complexity of patterns, but is more robust when making predictions on unseen data.

Optimal λ

The ideal value of λ is one that balances this trade-off. The goal is to find a λ that minimizes both bias and variance simultaneously, leading to the lowest possible *generalization error*. This can be done through methods such as cross-validation, where different values of λ are tested on validation sets to identify the one that results in the best performance on unseen data.

As λ increases, bias increases, but variance decreases. The ideal λ is the one where the sum of the bias and variance is minimized, providing the best generalization to new data.

- **Small** λ: Low bias, high variance (overfitting).
- **Large** λ: High bias, low variance (underfitting).
- **Optimal** λ: Balanced bias and variance, leading to a model that generalizes well to new data.

2 | Multi-Task Learning Model

2.1 | Problem Formulation

Task 1 (Regression): Predict the *SalePrice* of properties, y_{reg}, which is a continuous variable.

Task 2 (Classification): Classify properties into either *Premium* or *Standard* classes based on the *SalePrice*, y_{cls}, where $y_{\text{cls}} \in \{0, 1\}$, with 1 indicating "Premium" and 0 indicating "Standard".

2. Input Features

Let the feature vector for a property be $\mathbf{x} \in \mathbb{R}^d$, where d is the number of features (such as *OverallQual*, *YearBuilt*, etc.). These features are shared across both tasks.

Multi-Task Learning Model

The model aims to jointly learn the regression task $f_{\text{reg}}(\mathbf{x})$ and the classification task $f_{\text{cls}}(\mathbf{x})$. We use feature sharing to improve the performance of both tasks.

3. Joint Loss Function

We define a joint loss function L that combines both the regression loss L_{reg} and the classification loss L_{cls}:

$$L(\mathbf{w}, \mathbf{b}) = \alpha \cdot L_{\text{reg}}(\mathbf{w}_{\text{reg}}, \mathbf{b}_{\text{reg}}) + \beta \cdot L_{\text{cls}}(\mathbf{w}_{\text{cls}}, \mathbf{b}_{\text{cls}})$$

4. Regression Loss (Task 1: SalePrice Prediction)

For the regression task, we use the *Mean Squared Error (MSE)* as the loss function:

$$L_{\text{reg}}(\mathbf{w}_{\text{reg}}, \mathbf{b}_{\text{reg}}) = \frac{1}{n} \sum_{i=1}^{n} (y_{\text{reg},i} - f_{\text{reg}}(\mathbf{x}_i))^2$$

Where $y_{\text{reg},i}$ is the actual sale price of the i-th property and $f_{\text{reg}}(\mathbf{x}_i)$ is the predicted sale price.

5. Classification Loss (Task 2: Premium vs Standard Classification)

For the classification task, we use *Binary Cross-Entropy (BCE)* loss:

$$L_{\text{cls}}(\mathbf{w}_{\text{cls}}, \mathbf{b}_{\text{cls}}) = -\frac{1}{n}\sum_{i=1}^{n}[y_{\text{cls},i}\log(f_{\text{cls}}(\mathbf{x}_i)) + (1 - y_{\text{cls},i})\log(1 - f_{\text{cls}}(\mathbf{x}_i))]$$

Where $y_{\text{cls},i}$ is the true class (0 or 1) of the i-th property, and $f_{\text{cls}}(\mathbf{x}_i)$ is the predicted probability of the property being Premium.

2.1.1 | Why is multi-task learning suitable for this problem?

Multi-task learning (MTL) is suitable for this problem when multiple related tasks need to be solved simultaneously.
Multi-task learning (MTL) is well-suited for this problem because we have two related tasks: predicting SalePrice (regression) and classifying properties into Premium or Standard categories (classification). Both tasks share a common set of features such as OverallQual, Neighborhood, and Condition1, which impact both price prediction and classification. By learning these tasks simultaneously, the model can leverage shared representations between tasks, which can lead to better generalization and performance for both tasks compared to learning them separately.

- **Shared Representation**: model can learn better feature representations that are more generalizable across tasks, rather than overfitting to a single task.
- **Data Efficiency**: Training both tasks together allows the model to learn from the richer data source and improve performance on the data-scarce task.
- **Regularization Effect**:Model avoids overfitting to a single task, acting as a regularizer, beneficial when individual tasks may have noisy or limited data.

2.1.2 | How does feature sharing between tasks improve model performance?

In MTL, feature sharing refers to the practice of learning shared parameters or layers between different tasks.
For example, a high OverallQual might correlate with both higher SalePrice and a classification of "Premium." By sharing these features, the model can reduce the risk of overfitting and improve performance by learning joint representations that apply to multiple tasks. This helps to transfer knowledge between tasks and makes the model more robust.
This improves performance in the following ways:

- **Implicit Data Augmentation**:Model sees diverse variations of data, leading to a more robust feature space that helps with the generalization of each task.
- **Task-Specific Learning**: Task-specific components that fine-tune the shared representations for better performance
- **Reduced Risk of Overfitting**: Forcing the model to learn more general and robust features, rather than memorizing the specifics of one task.

2.2 | Data Understanding

2.2.1 | Compute the correlation

Numerical Values correlation matrix

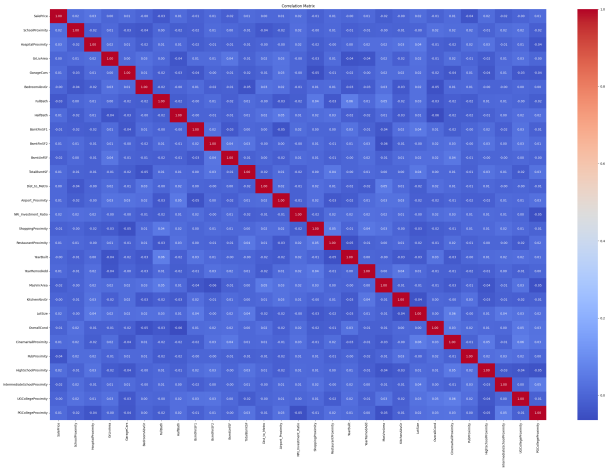


Figure 2.1: Correlation Matrix

2.2.2 | Handle missing values in YearBuilt using imputation techniques

Missing values in '*YearBuilt*' column handled using median imputation.

2.3 | Model Selection

2.3.1 | Justify using linear regression + logistic regression vs. decision trees

Linear Regression + Logistic Regression:

- Linear regression is suitable for continuous target variables like `SalePrice` because it models a linear relationship between features and the target.
- Logistic regression is appropriate for binary classification tasks like classifying properties as "Premium" or "Standard."
- Both models are easy to interpret and computationally efficient. They perform well when the relationship between the input features and the target is approximately linear.

Decision Trees:

- Decision trees can handle both regression and classification tasks and do not assume a linear relationship between the features and the target.
- They can capture non-linear relationships and interactions between features, but they are prone to overfitting, especially with small datasets.

2.3.2 | Discuss the role of feature engineering in improving model accuracy

Feature engineering plays a crucial role in improving model accuracy by creating new features that better capture the underlying patterns in the data. For example, interaction terms like *YearBuilt* × *OverallQual* can help capture the effect of older, high-quality properties on both price and classification.

3 | Data and Statistical Understanding

3.1 | Predict the 25th, 50th, and 75th quantiles of *SalePrice* in Mumbai

- 25th percentile of SalePrice in Mumbai: 8253150.00
- 50th percentile (Median) of SalePrice in Mumbai: 10290847.00
- 75th percentile of SalePrice in Mumbai: 12840374.00

Higher median SalePrice detected. Setting alpha=0.5, beta=0.5, lambda=0.1

3.2 | Analyze the distribution of *SalePrice*

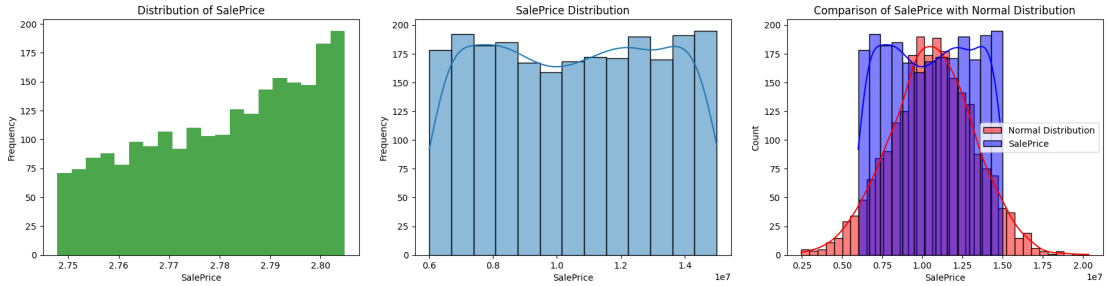


Figure 3.1: SalePrice Target Analysis

Skewness of SalePrice: -0.0051. Plots are shown in fig 3.1

3.3 | Define the hypothesis space for both regression and classification tasks

- **Regression (SalePrice)**: The hypothesis space for linear regression is the set of all linear functions that model the relationship between the input features and the target variable `SalePrice`. For decision trees, the hypothesis space consists of all possible tree structures.
- **Classification (Premium vs Standard)**: For logistic regression, the hypothesis space is the set of all sigmoid functions mapping the input features to a binary outcome. For decision trees, it includes all possible decision rules for splitting the data.

3.4 | Explain how the choice of hypothesis space affects model performance

The choice of hypothesis space directly affects model performance because it defines the model's capacity to fit the data:

- A too-simple hypothesis space (e.g., linear functions) may lead to underfitting, where the model cannot capture complex patterns in the data.
- A too-complex hypothesis space (e.g., deep decision trees) may lead to overfitting, where the model captures noise rather than generalizable patterns.

3.5 | Modify hyperparameters based on quantile analysis

Based on the quantile analysis, you can adjust the regularization parameters to improve model performance:

- α controls the weight of the regression loss (SalePrice).
- β controls the weight of the classification loss (Premium vs Standard).
- λ controls the strength of regularization.

For properties in the lower quantiles (e.g., 25th), we may reduce α to focus more on classification accuracy (β). For higher quantiles, we can increase α to emphasize price prediction.

4 | Analysis

In this section, the analysis as per guidelines is done:

4.1 | Analysis of Optimization Techniques

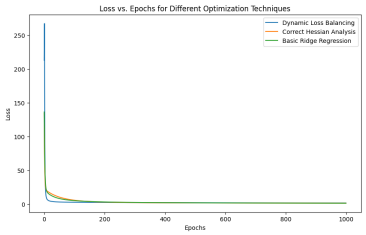


Figure 4.1: Comparison of Different Optimization Techniques

Model	Test Loss	MSE	BCE	Regularization Loss
Dynamic Loss Balancing	1.8192	1.9183	1.7851	0.0340
Correct Hessian Analysis	1.6120	1.9495	2.0101	0.0342
Basic Ridge Regression	1.6411	1.9992	1.9931	0.0436

Table 4.1: Evaluation Results of Different Models

4.2 | Analysis of Regularization Techniques

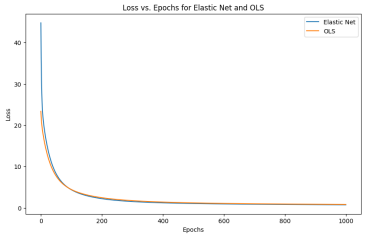


Figure 4.2: Comparison of Different Regularization Techniques

Model	Test Loss	MSE	BCE	Regularization Loss
Elastic Net	12.3531	23.4288	1.8811	0.0744
OLS (Ordinary Least Squares)	12.3510	23.2391	2.1736	0.0794

Table 4.2: Evaluation Results of Elastic Net and OLS Models

4.3 | Analysis of Feature Selection Techniques

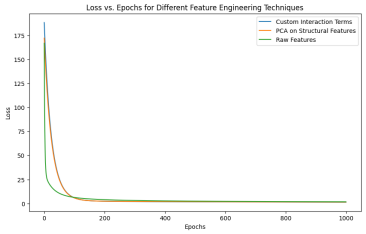


Figure 4.3: Comparison of Different Feature Selection Techniques

Model	Test Loss	MSE	BCE	Regularization Loss
Interaction Terms	1.3663	0.0672	2.5701	0.0477
PCA	1.0602	0.0661	1.9328	0.0608
Raw Features	1.7839	1.8559	1.5464	0.0827

Table 4.3: Evaluation Results of Interaction Terms, PCA, and Raw Features Models

4.4 | Feature importance plots for OverallQual and YearBuilt

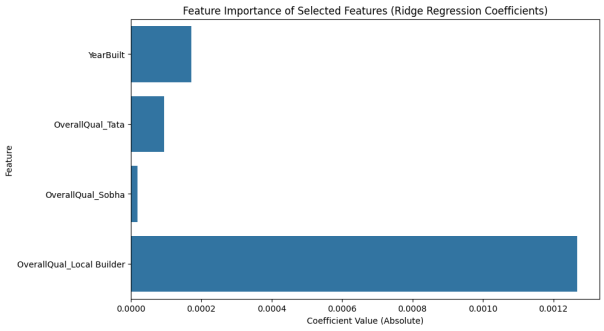


Figure 4.4: Feature Importance

4.5 | Regularization path analysis forλ

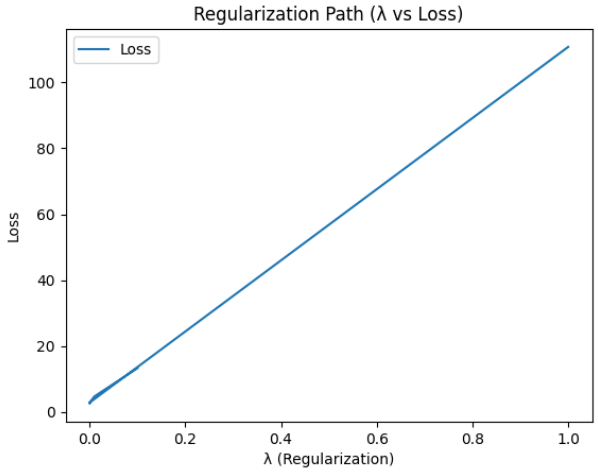
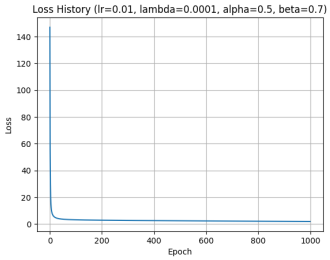
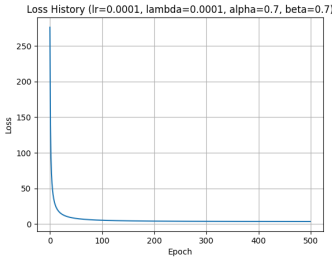
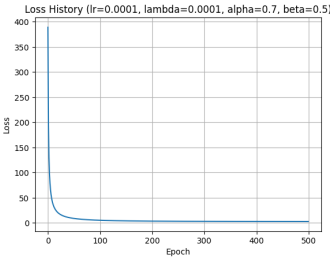


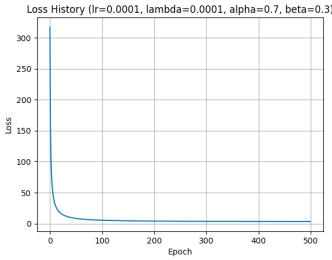
Figure 4.5: Regularization path analysis for 500 epochs

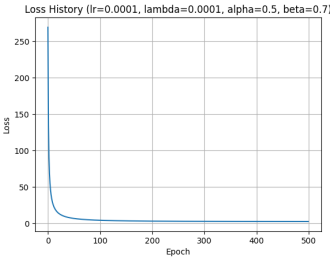
5 | Hyperparamter Tuning Plots for 500 epochs

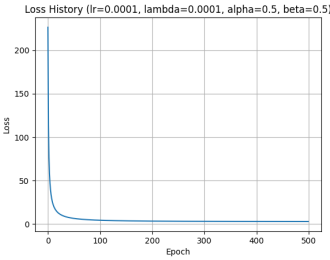


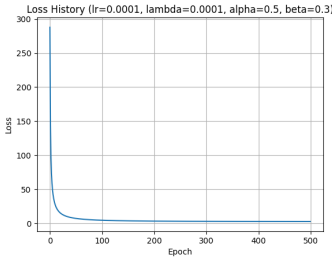


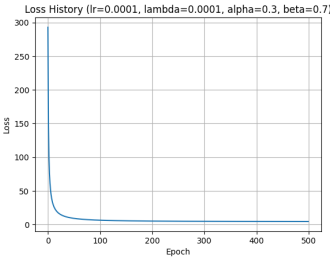


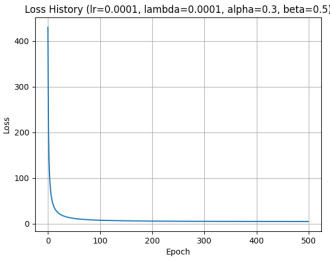


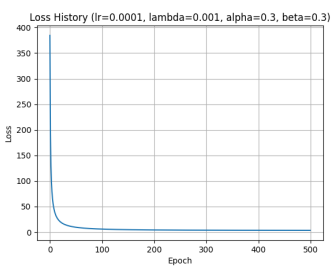


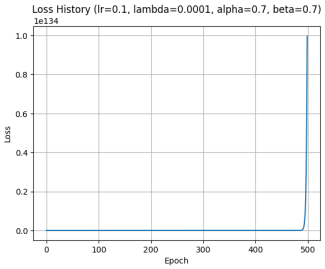


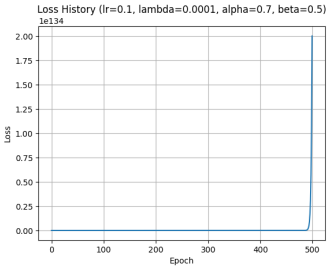


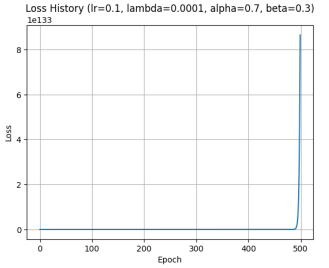


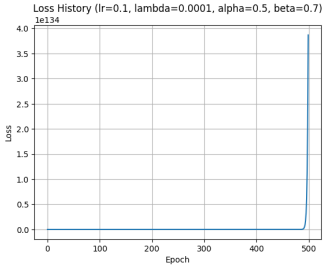


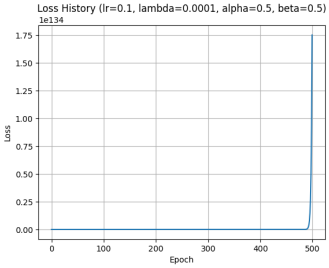












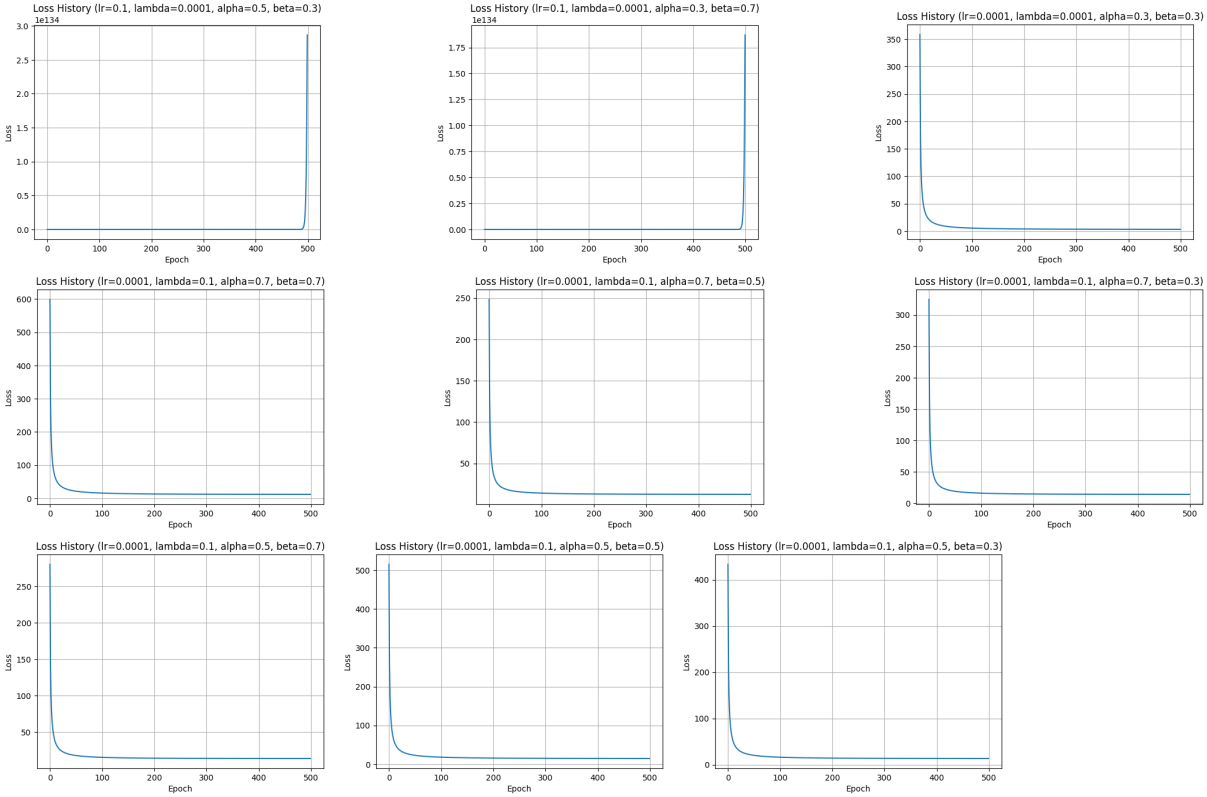


Figure 5.1: Finding Best Hyperparameters

Metric	Value
Training Loss	2.5884
Regression Metrics:	
Mean Squared Error (MSE)	10.0015
Classification Metrics:	
Accuracy	0.4793
Precision	0.4623
Recall	0.5288
F1-Score	0.4933

Table 5.1: Training Loss, Regression Metrics, and Classification Metrics

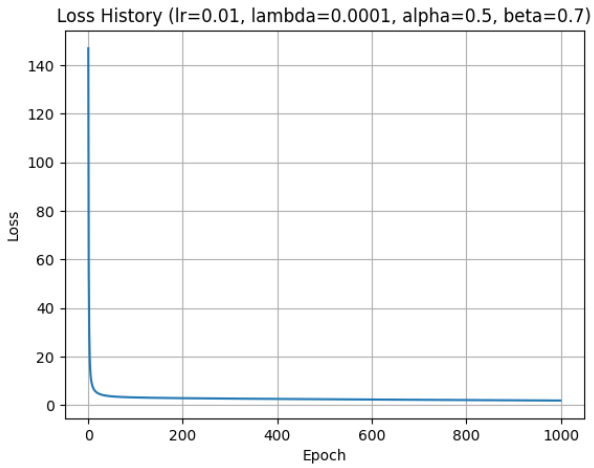


Figure 5.2: Model Loss vs Epochs for Best Hyperparameter Combination