CS7.403: Statistical Methods in AI
Assignment-1

Chetan Mahipal

Course Instructor - Dr. Vineet Gandhi

IIIT Hyderabad - February 13, 2025

1 Predicting Food Delivery Time

1.1 Exploratory Data Analysis

The following plots were obtained on exploring the dataset

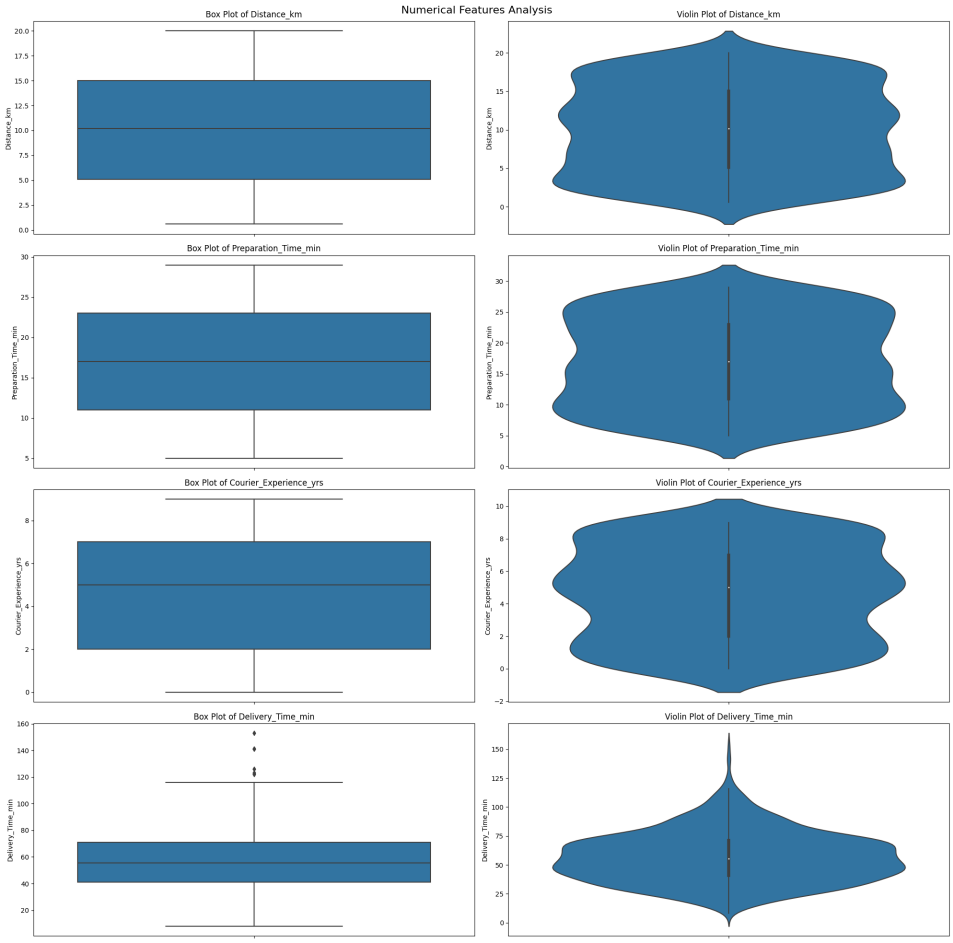

Figure 1: *Numerical Features*

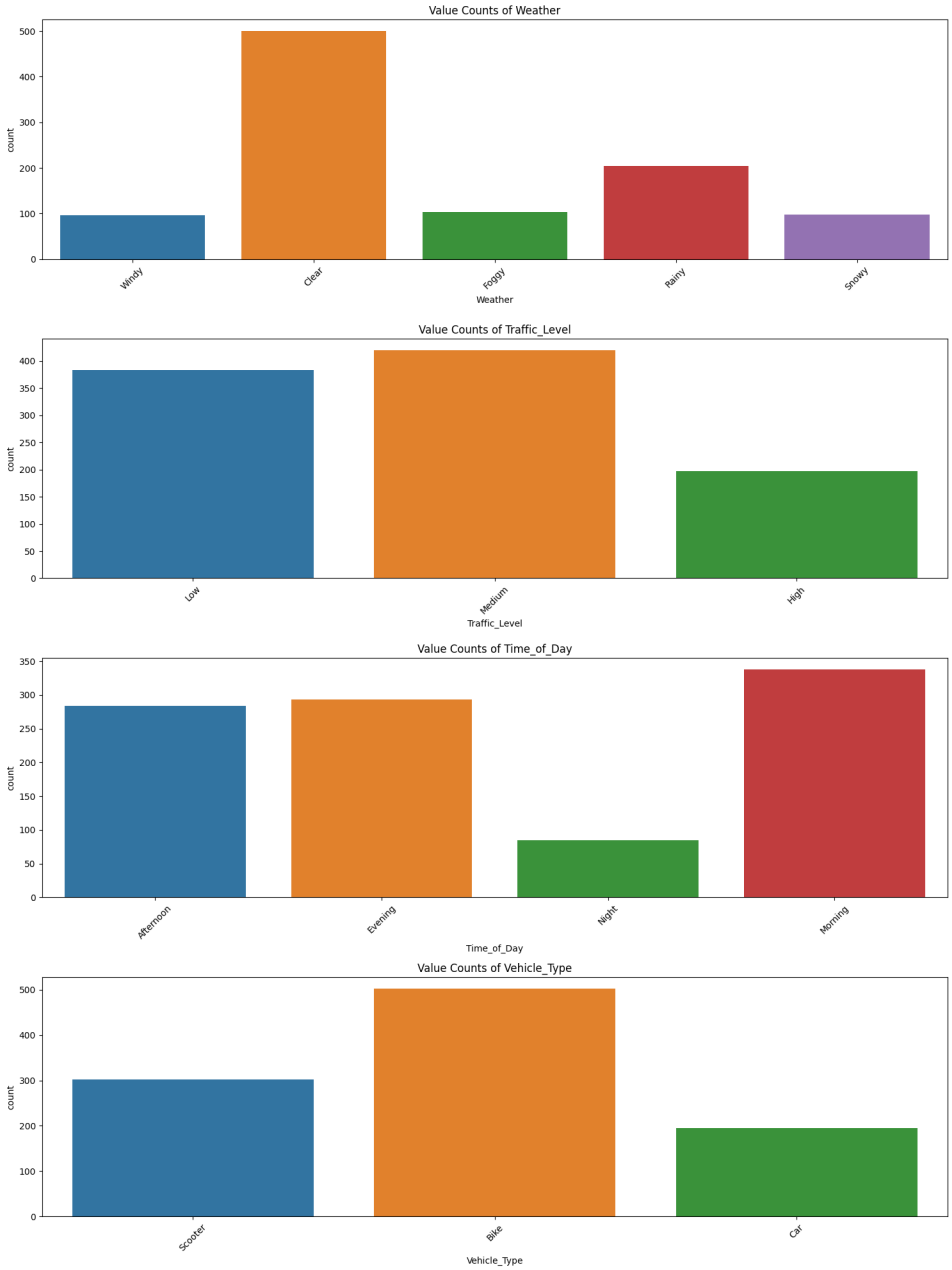

Figure 2: *Categorical Features*

Observations & Insights

- Delivery time is highly variable, likely influenced by distance, traffic, and weather.
- High traffic levels and poor weather conditions may contribute to longer delivery times.
- Morning and evening have the highest order volumes, indicating peak hours.
- The dominance of bikes and scooters suggests an emphasis on quick, urban deliveries.

1.2 Linear Regression with Gradient Descent

This report evaluates the performance of different gradient descent techniques—Batch, Mini-Batch, and Stochastic—in training a linear regression model. The goal is to analyze their impact on test mean squared error (MSE) and R^2 score.

Method	Test MSE	R^2 Score
Batch	68.3127	0.8355
Mini − Batch	72.5156	0.8253
Stochastic	103.0410	0.7518

The following plots were generated on 1000 iterations of training:

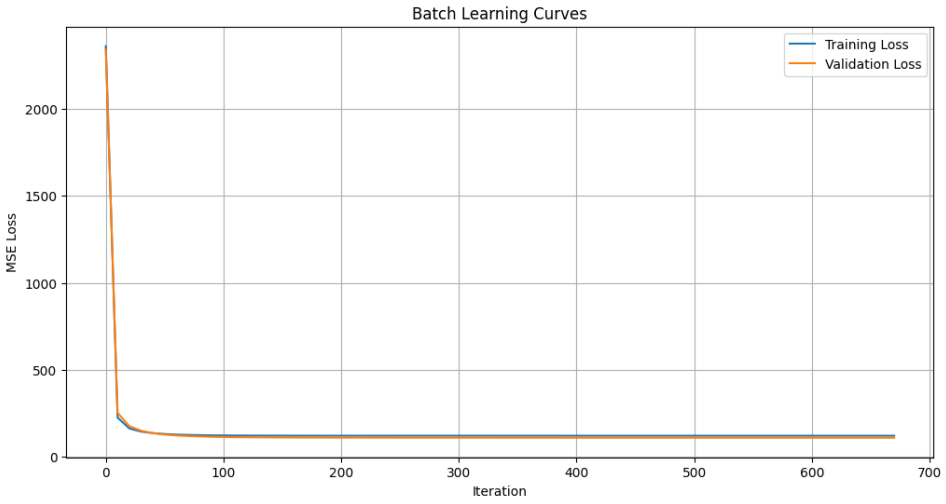

Figure 3: *Batch Gradient Descent*

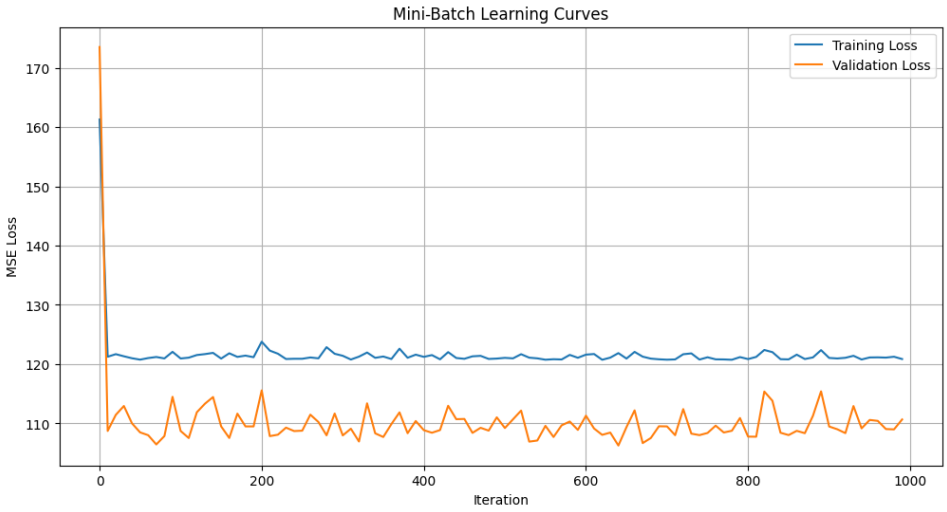

Figure 4: *Mini-Batch Gradient Descent*

1.3 Regularization

This section presents the evaluation of Lasso and Ridge regression models, comparing their performance in terms of Test MSE, R^2 Score, and the average absolute coefficient value.

1.3.1 Lasso Regression Results

Lasso regression was trained and evaluated, with the following performance metrics:

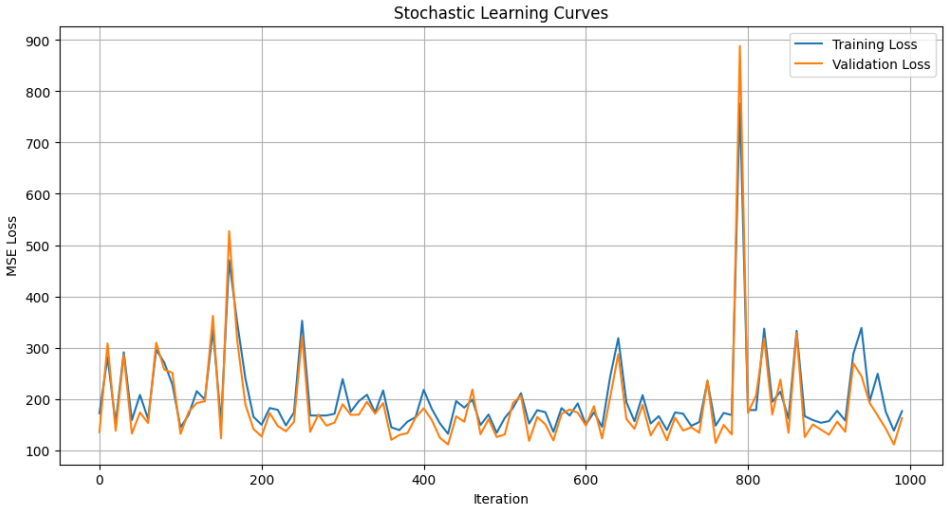

Figure 5: *Stochastic Gradient Descent*

Metric	Value
Test MSE	83.3273
R^2 Score	0.7993
Average Absolute Coefficient	3.0547

Table 1: *Performance metrics of Lasso regression.*

1.3.2 Ridge Regression Results

Similarly, Ridge regression was trained and evaluated:

Metric	Value
Test MSE	134.0922
R^2 Score	0.6770
Average Absolute Coefficient	2.1201

Table 2: *Performance metrics of Ridge regression.*

1.3.3 Comparison Summary

The table below summarizes the comparative performance of Lasso and Ridge regression:

Method	Test MSE	R^2 Score
Ridge	134.0922	0.6770
Lasso	83.3273	0.7993

Table 3: *Comparison of Ridge and Lasso regression performance.*

The following plots are obtained for Ridge and Lasso:

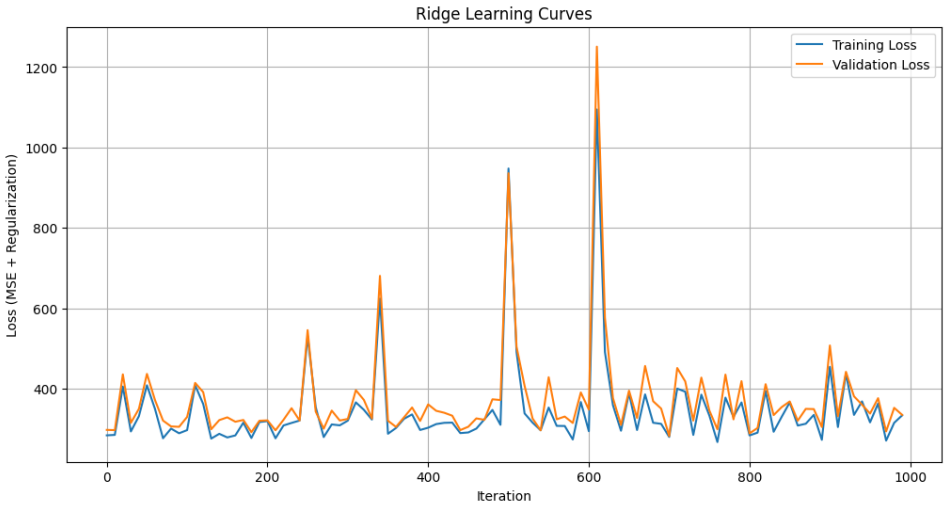

Figure 6: *Ridge Regularization*

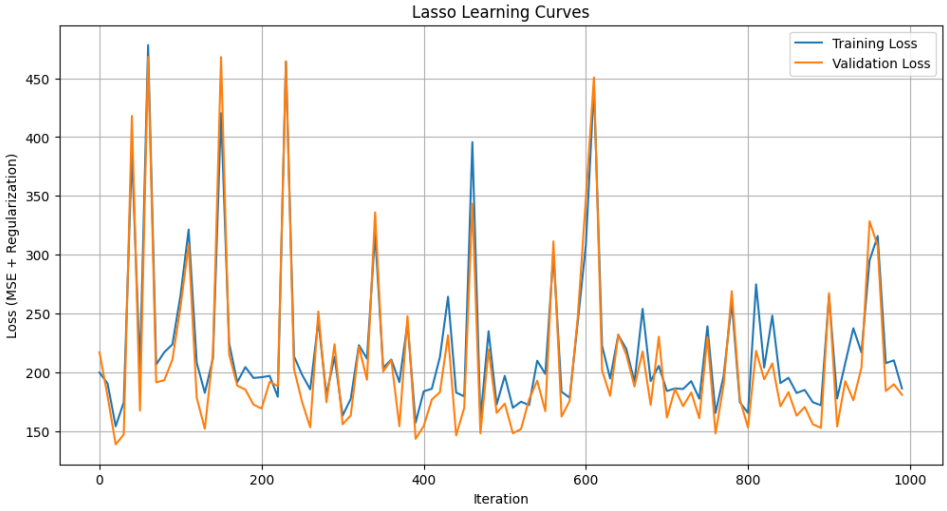

Figure 7: *Lasso Regularization*

The optimal values of λ for both Ridge and Lasso regression are as follows:

Method	**Optimal λ**	**Minimum Test MSE**
Ridge	0.2222	83.8957
Lasso	0.5556	74.5140

Table 4: *Optimal regularization parameters and minimum test MSE values for Ridge and Lasso regression.*

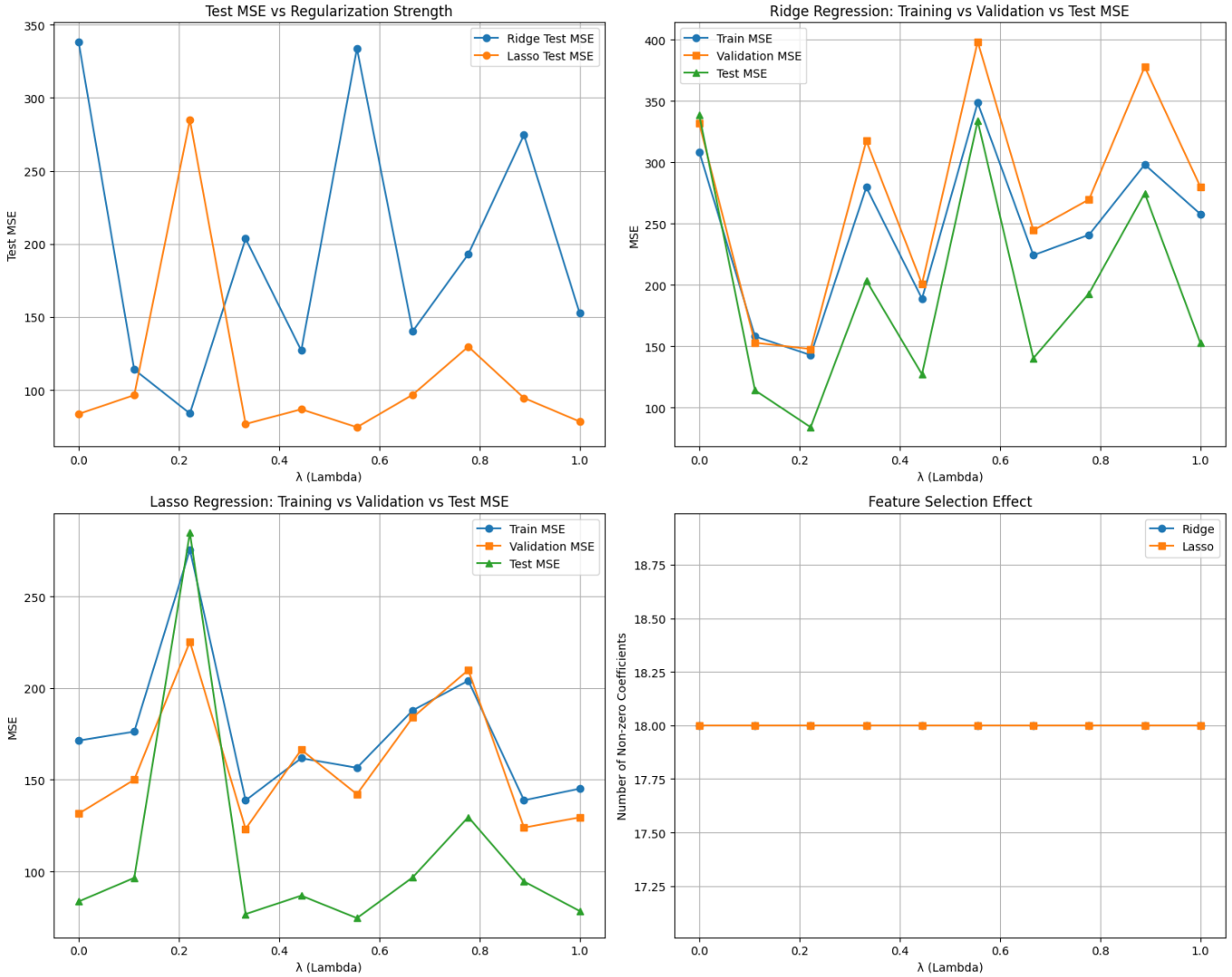

Figure 8: *Optimal regularization parameters*

1.4 Report

1.4.1 Difference between Types of GDA

1. Batch Gradient Descent
Description:

- Computes the gradient using the entire dataset.
- Updates weights after processing all training samples.

Advantages:

- Converges smoothly due to stable updates.
- More efficient for convex loss functions.

Disadvantages:

- Slow for large datasets due to full dataset computation.
- Can get stuck in local minima.

2. Stochastic Gradient Descent (SGD)
Description:

- Updates weights after computing the gradient for a single training sample.

Advantages:

- Faster updates, allowing quicker learning.
- Can escape local minima due to randomness.

Disadvantages:

- High variance in updates, leading to instability.
- Noisy convergence, requiring learning rate decay.

3. Mini-Batch Gradient Descent
Description:

- Computes the gradient on a small batch of samples before updating weights.

Advantages:

- Balances stability and efficiency.
- Uses vectorized operations, improving computation speed.

Disadvantages:

- Still has some variance in updates compared to batch GD.
- Requires tuning batch size for optimal performance.

Which Converges the Fastest?

- **Stochastic Gradient Descent (SGD)** generally converges the fastest in terms of iteration count but may be noisy.
- **Mini-Batch Gradient Descent** offers a good balance between speed and stability, often preferred in deep learning.
- **Batch Gradient Descent** is the slowest but most stable.

1.4.2 Regularization Impact on Model Performance

Comparison Summary:

Method	Test MSE	R^2 Score
Ridge	134.0922	0.6770
Lasso	83.3273	0.7993

Table 5: *Comparison of Ridge and Lasso regression performance for* $\lambda = 0.5$.

Optimal Regularization Parameters:

- **Ridge optimal** λ: 0.2222

- **Lasso optimal** λ: 0.5556

Minimum Test MSE Values:

- **Ridge**: 83.8957
- **Lasso**: 74.5140

Influence of Regularization:

- **Ridge**: Helps reduce model complexity by shrinking coefficients, preventing overfitting while maintaining important features.
- **Lasso**: Performs feature selection by forcing some coefficients to zero, leading to a sparse model.

Optimal λ Selection:

- **Ridge**: $\lambda = 0.2222$ (yields the best test MSE)
- **Lasso**: $\lambda = 0.5556$ (yields the best test MSE)

Impact of Feature Scaling:

- **Standardization** is crucial for both Ridge and Lasso as these methods are sensitive to feature magnitudes.
- Scaling ensures that no single feature dominates due to large numeric ranges.
- Improves convergence and stability of gradient-based optimizers.

1.4.3 Analysis of Feature Weights in Regularized and Non-Regularized Models

To understand how regularization affects feature importance, we visualize the trained model weights for the best-performing Ridge and Lasso models compared to a non-regularized model using a bar plot. This helps in identifying the influence of each feature on delivery time.

Results:

- **Number of near-zero weights in Lasso**: 5
- **Features with nearly zero effect on delivery time**: [4, 7, 10, 14, 15]

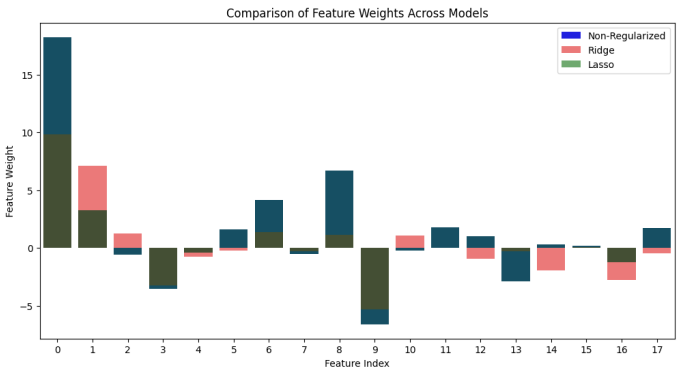

Figure 9: *Feature Weights*

2 KNN and ANN

2.1 Locally Sensitive Hashing

2.1.1 Histogram Analysis of Sample Distribution Across Buckets

To analyze how the number of hyperplanes affects the partitioning of data, we plot histograms showing the frequency of samples in each bucket for different hyperplane counts.

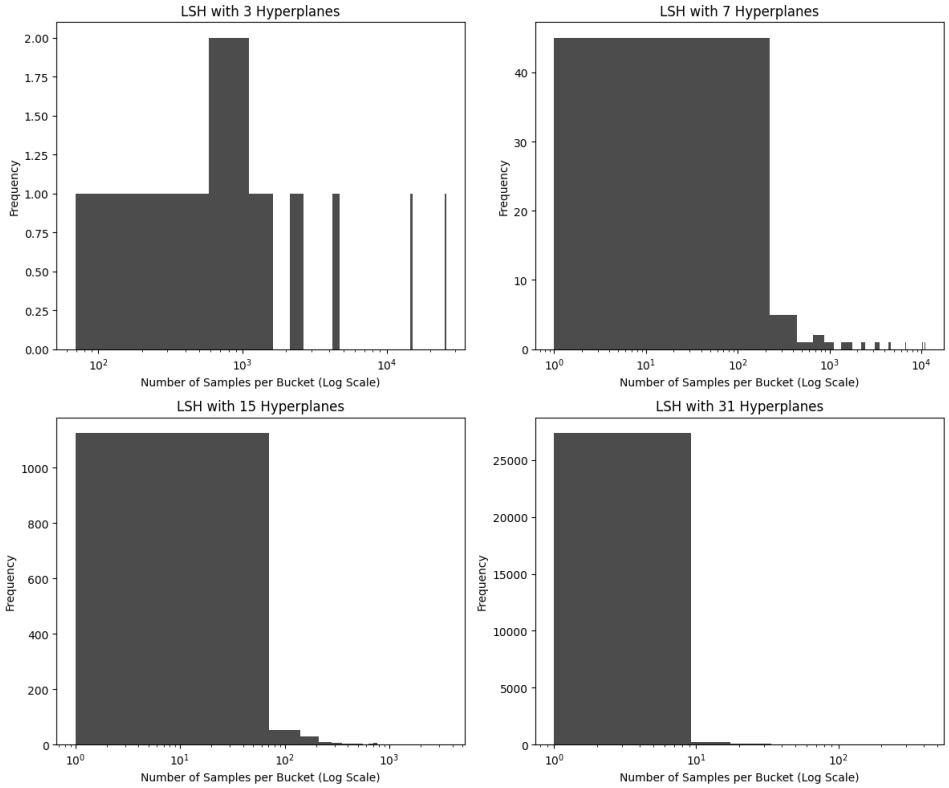

Figure 10: *Histograms showing sample distribution across buckets for varying numbers of hyperplanes.*

Problem Observed: Skewed Bucket Distributions

- The histograms for 15 and 31 hyperplanes are dominated by a single bar, indicating that most buckets contain only one or very few samples.
- This suggests that the number of hyperplanes is too high, causing excessive partitioning and leading to sparsely populated buckets.

2.1.2 Effect of Varying Number of Hyperplanes on Performance Metrics

To understand how the number of hyperplanes influences retrieval performance, we analyze key metrics such as Mean Reciprocal Rank (MRR), Precision@100, Hit Rate, and Average Comparisons Per Query.

- **Mean Reciprocal Rank (MRR)**
  - **Trend:** MRR initially **increases**, then **decreases** beyond an optimal hyperplane count.
  - **Reason:** More hyperplanes create finer partitions, improving rank ordering of retrieved neighbors. However, excessive partitioning may over-fragment data, leading to missing relevant neighbors.

- **Precision@100**
  - **Trend:** Precision@100 generally **increases** with more hyperplanes until it stabilizes or slightly declines.
  - **Reason:** More hyperplanes improve the quality of retrieved results by reducing false positives. However, excessive partitioning can cause relevant items to be assigned to different buckets, leading to missed results.
- **Hit Rate**
  - **Trend:** Hit Rate remains **high** but may decrease slightly for very high hyperplane numbers.
  - **Reason:** The system continues to find relevant neighbors, but extreme fragmentation causes some queries to miss relevant results, reducing hit rate marginally.
- **Average Comparisons Per Query**
  - **Trend:** Comparisons per query **decrease initially**, then **increase sharply** with more hyperplanes.
  - **Reason:**
    - Initially, LSH reduces search space, decreasing comparisons.
    - Excessive hyperplanes cause many small buckets, forcing a fallback to exhaustive search in some cases, increasing comparisons.

2.2 IVF

2.2.1 Analysis of Clustering and Query Comparisons

To analyze the clustering performance and query efficiency, we present the following plots:

- **Cluster Size Distribution:** A plot showing the number of points assigned to each cluster.
- **Comparisons vs. n_{probe}:** A plot illustrating how the average number of comparisons per query changes with different values of n_{probe}.

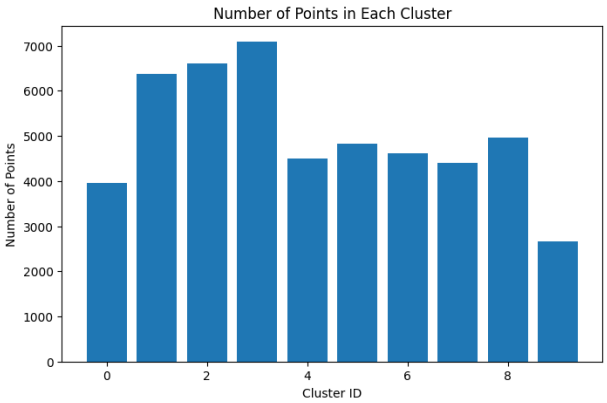

Figure 11: *Number of points in each cluster.*

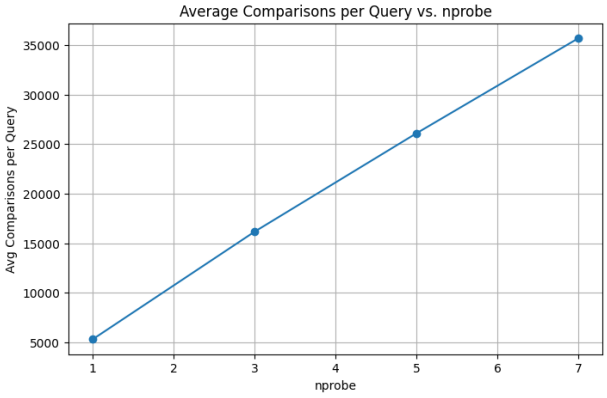

Figure 12: *Average number of comparisons per query vs.* n_{probe}.

2.3 Analysis

2.3.1 Comparison of Methods on a Common Task

To evaluate the performance of the three methods—IVF with $n_{probe} = 5$, LSH, and Normal KNN—we compare their Mean Reciprocal Rank (MRR), Precision@100, Hit Rate, and Average Comparisons Per Query.

Method	MRR	Precision@100	Hit Rate	Avg Comparisons/Query
IVF ($n_{probe} = 5$)	0.9347	0.8412	0.9995	26092.34
LSH	0.4685	0.2866	0.9894	1795
Normal KNN	0.9348	0.8411	0.9996	50000

Table 6: *Performance comparison of different methods.*

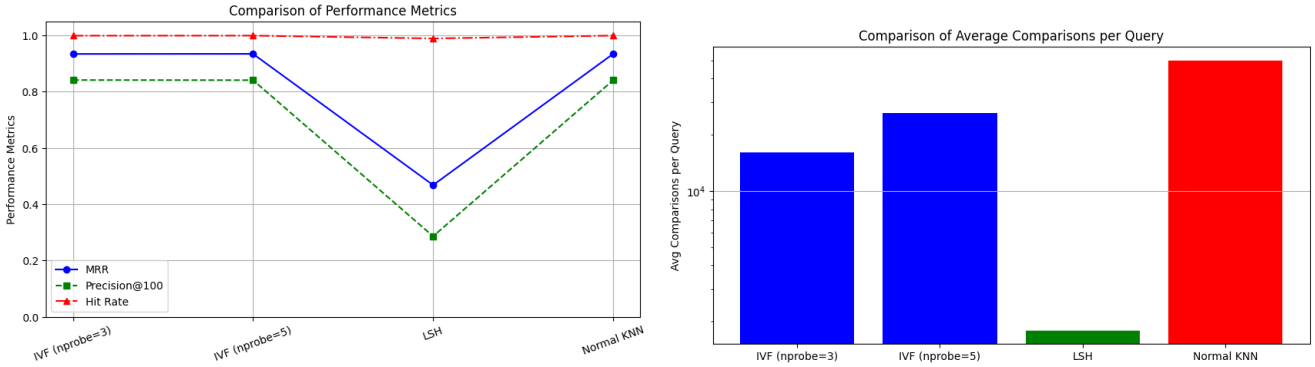

Figure 13: *Performance metrics across different methods.*

2.3.2 Explanation of Trends

- **Normal KNN** achieves the best performance in terms of MRR, Precision@100, and Hit Rate but requires the highest number of comparisons per query.
- **IVF** ($n_{probe} = 3, 5$) maintains high performance while significantly reducing the number of comparisons compared to Normal KNN. Increasing n_{probe} improves accuracy but increases computations.
- **LSH** is the most efficient in terms of comparisons but sacrifices significant accuracy, making it less suitable when high precision is required.

3 The Crypto Detective

3.1 Exploratory Data Analysis

To understand feature relationships and their impact on fraud detection, we analyze the correlation matrix for 13 numerical features and visualize feature distributions using violin plots. The correlation matrix reveals dependencies between variables, while violin plots highlight variations in feature values based on the FLAG column.

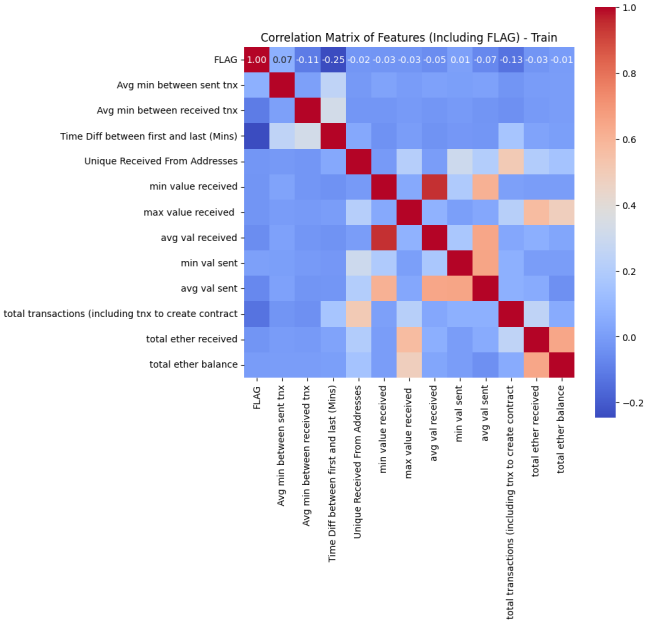

Figure 14: *Confusion Matrix for Co-relation*

1. FLAG Correlation with Other Features

- The `FLAG` variable (indicating fraud or anomaly) has relatively low correlations with most features.
- The highest correlation (negative) appears to be with **"Time Diff between first and last (Mins)"** (≈ -0.25), suggesting that fraudulent or flagged accounts tend to have transactions occurring over a shorter duration.
- There is a small positive correlation with **"Avg min between sent tnx"** (≈ 0.07), meaning flagged accounts might have slightly larger gaps between sent transactions.

2. Relationships Among Transaction Features

- **"Total transactions (including tnx to create contract)"** has a strong positive correlation with **"Total ether received"** and **"Total ether balance"**, which is expected as more transactions typically result in higher balances.
- **"Min value received"**, **"Max value received"**, and **"Avg value received"** are positively correlated with each other, indicating that accounts receiving large transactions tend to receive higher average amounts.

3. Weak Correlations Across Features

- Most correlations are weak, meaning individual features may not strongly influence each other.
- This suggests that fraud detection (`FLAG`) is not heavily dependent on any single variable.

3.2 Hyper-Paramter Tuning

We explore key decision tree hyperparameters, including **maximum tree depth** and **minimum samples per split**, to understand their effect on model performance. Additionally, we compare **entropy** and **Gini impurity** as splitting criteria, using validation accuracy to assess their impact.

3.2.1 Performance Analysis

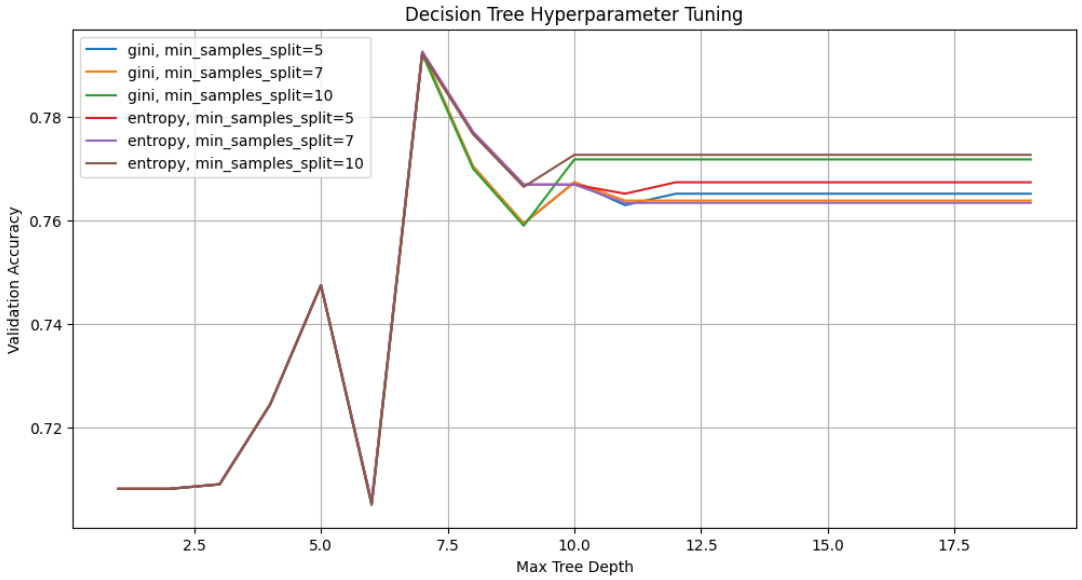

Figure 15: *Decision Tree Accuracy vs. Hyperparameters*

3.2.2 Observations

1. **Max Depth Tradeoff:** Increasing depth improves training accuracy but eventually harms validation accuracy due to overfitting.
2. **Min Samples Split Effect:** Larger values prevent small, noisy splits, enhancing generalization. At depth = 10 or more, **num_split = 10** performs better than others.
3. **Entropy vs. Gini:** The optimal criterion depends on the dataset distribution and how well **information gain (entropy)** separates classes. In our dataset, **entropy performs better**.

3.3 Model Visualization

Feature importance analysis helps identify the most influential predictors, while decision tree visualization provides insight into the model's decision-making process.

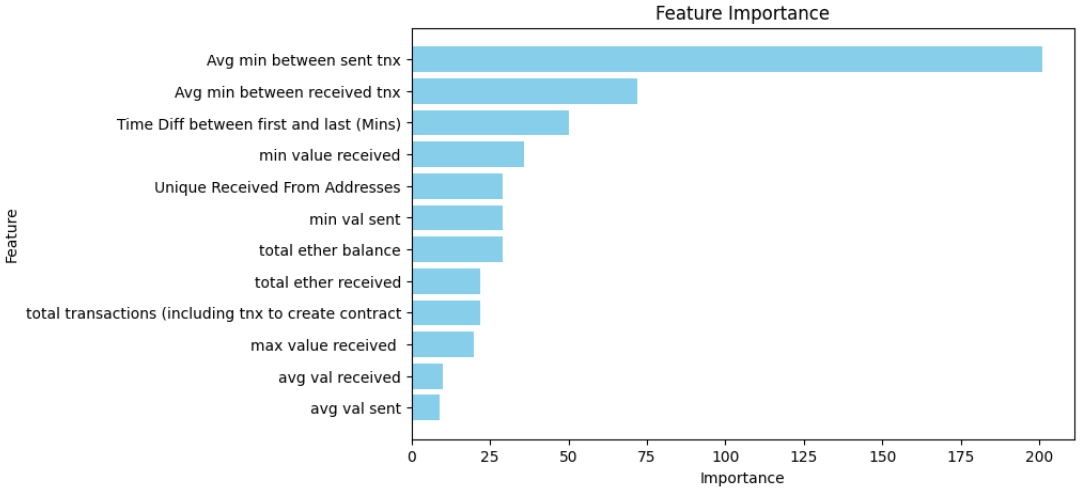

Figure 16: *Feature importance visualization highlighting key predictors.*

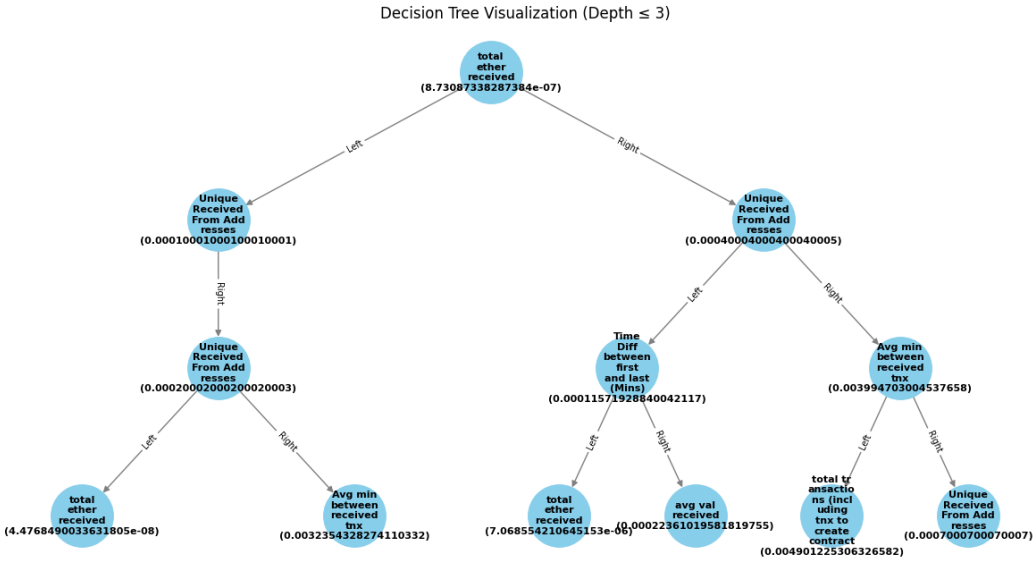

Figure 17: *Graphical representation of the decision tree structure.*

4 K-Means

4.1 Effect of Hyper-Parameters

Superpixel Count (K)

- Increasing K results in more, smaller superpixels, which better adhere to object boundaries.
- Decreasing K results in fewer, larger superpixels, leading to coarser segmentation.

Compactness (m)

- Higher m (e.g., $m = 40$) results in more compact, regularly shaped superpixels, making them less sensitive to image edges.
- Lower m (e.g., $m = 20$) allows superpixels to follow image edges more closely, leading to more irregular shapes but better boundary adherence.

4.2 Effect of RGB as Metric

Differences Between Lab and RGB Space in SLIC

- **Lab Space:**
  - Produces well-defined superpixels.
  - Preserves object boundaries.
  - Uses the L channel for luminance and a/b channels for color differences, making segmentation perceptually accurate.
- **RGB Space:**
  - Results in blurry, less distinct superpixels.
  - Poor boundary adherence.
  - Euclidean distance in RGB does not align with human visual perception, leading to improper clustering of colors.

Why RGB Results in Blurry Superpixels

- **Perceptual Uniformity:**
  - RGB treats all three channels equally, while human vision is more sensitive to luminance.
  - Lab space better mimics this perception, improving segmentation.
- **Distance Metric Distortion:**
  - Euclidean distances in RGB are not perceptually uniform.
  - Causes merging of numerically similar but visually distinct colors.
- **Edge Preservation Issue:**
  - Lab space considers luminance separately, maintaining details in shadows and highlights.
  - RGB space merges areas indiscriminately, leading to loss of structure.

4.3 Optimization

Optimization Method Used

- The optimization method involves reducing the number of iterations dynamically using a decay function.
- The number of iterations per frame follows:
  - **Iterations decrease progressively** from 10 down to a minimum of 3, ensuring computational efficiency.
  - The decay rate applied is **0.9**, meaning each frame uses fewer iterations than the previous one.

Average Number of Iterations per Frame

- **Optimized Case:**
  - Processed frames with varying iterations: **[10, 9, 7, 5, 3, 3, 3, 3, 3, 3, 3]**
  - **Average iterations per frame:** ≈ 5.27
- **Non-optimized Case:**
  - **Fixed at 10 iterations per frame.**
  - **Average iterations per frame: 10**

Observations

- The optimized approach **reduces computational cost** while maintaining segmentation quality.
- Later frames require fewer iterations, indicating the algorithm's convergence.
- A decay-based iteration schedule balances speed and accuracy.