SMAI-Asn1 report

Akshara S Bhat

February 2025

1 Question 1

1.1 Three types of gradient descent algorithms

Batch gradient descent
Calculates the gradient based on the entire training dataset at each iteration, updating the model parameters only after processing all data points.
Often preferred for smaller datasets where computing the gradient over the whole set is manageable.
Advantages
1.Provides stable convergence.
2. Has less MSE.
Disadvantages
1.Can be computationally expensive for large datasets as it requires processing all data points in each iteration.
2.May get stuck in local minima if the loss function is non-convex.

Mini Batch gradient descent
The data set is divided into different batches and a one of those batches is randomly picked in an iteration for updating the weights.
It is like an intermediate of Batch and stochastic gradient descent

Advantage
1.Provides a trade-off between the computational efficiency of stochastic gradient descent and the stability of batch gradient descent.

2.Provides faster convergence compared to the stochastic gradient(in terms of number of iterations) descent while reducing the variance in updates.

Disadvantages
1.May still get stuck in local minima or saddle points, especially in non-convex loss functions.
2.Selecting the batch size can be challenging; too small of a batch size increases noise, while too large can be computationally expensive.

Batch gradient converged fastest in terms of iterations but, stochastic gradient descent took lesser time overall

1.2 Lasso and Ridge regularization

Lasso and Ridge regularization increased MSE(Lasso had lesser MSE than Ridge). The optimal λ turned out to be 0. This is because the data are already simple and don't have much noise, so it is not over-fitting.

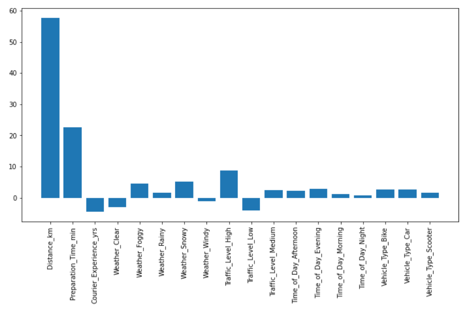

Figure 1: non-regularized weights

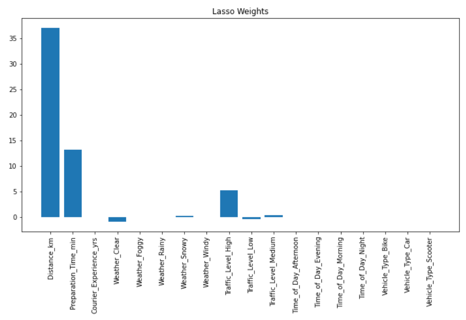

Figure 2: lasso weights

The features *Distance*, *Preparation time* were the most important in determining the Delivery time. Traffic level and weather were other significant features. Lasso reduced most of the features to zero or near zero. Ridge reduced the weights for the features but did not reduce them to zero.

1.3 Affect of scaling

If features have different scales (e.g., one feature ranges from 0 to 1, while another ranges from 1000 to 10000), the gradient descent updates will be dominated by the feature with larger values. This leads to slower convergence or a less efficient search for the optimal parameters.

1.4 Exploratory Data Analysis (EDA)

Observations:

- Distance has is fairly normal distribution
- Courier Experience has is right skewed a bit. It has peaks at intervals, probably because it is measured in years.
- Preparation time has lot of outliers to the right.(higher time)
- Weather is clear in most cases and traffic is moderate or low, less deliveries in the night and mostly delivered by bikes
- As distance increases Delivery time also increases

On comparing random and zero initialisation of weights, I found that, random initialisation leads to faster convergence in stochastic gradient descent, but the MSE was lower in zero initialisation(probably because the true weights are near zero). Therefore I used zero initialization.

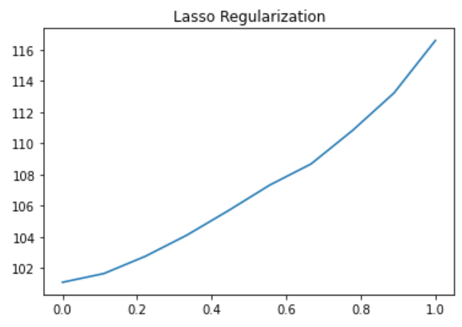

Figure 3: mse vs lasso

MSE and R-2 score
batch mse: 72.06130236186358
batch r2 score: 0.7588779841231409
mini batch mse: 104.8482495288279
mini batch r2 score: 0.6491706302975406

stochastic mse: 114.99150742608795
stochastic r2 score: 0.6152305999125145

Regularization performance, λ=5

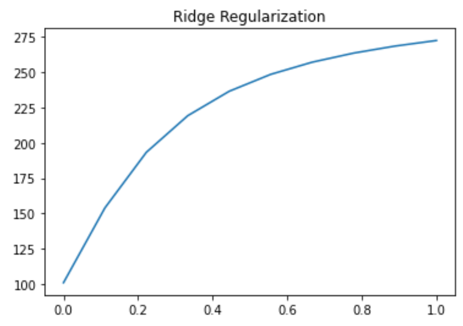

Figure 4: mse vs regularization strength

ridge mse: 243.00081712787195
lasso mse: 106.48982822242057

Analysis

Lasso has comaparitively lesser impact and has accelerating imapct as λ increases and the vice-versa is true for ridge. The model hasn't overfit and both regularizations are making the model underfit.

2 Question 2

Note: The dataset shall be in the same format as given in the assignment document's link and shall be in a directory called *embeddings*

2.1 KNN

Accuracy for cosine dist for k=1: 0.9048
Accuracy for euclidean dist for k=1: 0.9048
Accuracy for cosine dist for k=5: 0.9181
Accuracy for euclidean dist for k=5: 0.9182
Accuracy for cosine dist for k=10: 0.9194
Accuracy for euclidean dist for k=10: 0.9194

Accuracy for text embeddings with cosine distance: 0.8781
Accuracy for text embeddings with euclidean distance: 0.8781

```
Mean reciprocal rank for text embeddings with euclidean distance: 1.0
Precision for class 0: 1.0
Precision for class 1: 0.96
Precision for class 2: 1.0
Precision for class 3: 0.92
Precision for class 4: 1.0
Precision for class 5: 0.9
Precision for class 6: 0.99
Precision for class 7: 1.0
Precision for class 8: 1.0
Precision for class 9: 0.97
Hit rate for text embeddings with euclidean distance: 1.0
mean Precision @100 for image to image retrieval: 0.8410829999999664
mean Hit rate for image to image retrieval: 0.9996
mean reciprocal rank for image to image retrieval: 0.9347961513315047
```

2.2 LSH

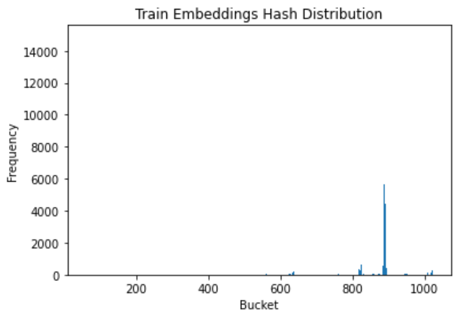

Figure 5: number of hyperplanes = 10

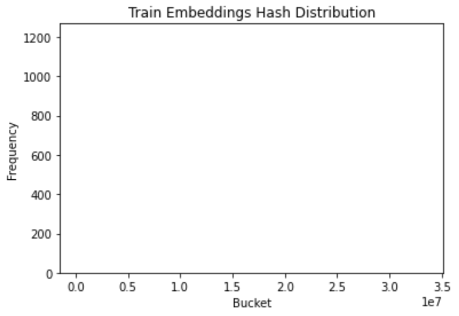

Figure 6: number of hyperplanes = 25

As we increase the number of hyperplanes, the number of images in each bucket becomes smaller and smaller. Since most of the vectors are concentrated in a small region, it becomes difficult to visualize and the effectiveness of increasing the number of hyperplanes also decreases.

Metrics for number of hyperplanes = 25
Accuracy for euclidean dist for k=5: 0.6713
Mean Reciprocal Rank with lsh: 0.7300272307573475
Mean Precision @ 100 with lsh: 0.1873679999999934
Hit Rate @ 100 with lsh: 0.8117

Metrics for number of hyperplanes = 10
Accuracy for euclidean dist for k=5: 0.8882
Mean Reciprocal Rank with lsh: 0.9139990724149184
Mean Precision @ 100 with lsh: 0.7098769999999807
Hit Rate @ 100 with lsh: 0.9926

Evidently the model with number of hyperplanes 10 has better performance, this is because increasing the number of hyperplanes reduces the size of each bucket, leading to sparser data distribution, which reduces the model's overall effectiveness.

2.3 IVF

nprobe	MRR	Precision@100	Hit Rate@100
1	0.927	0.835	0.999
3	0.934	0.842	0.999
5	0.935	0.841	1.000

2.4 Comparison

Task: Text to Image Retrieval

For the task of text to image retrieval, we compare the performance of three methods: K-Nearest Neighbors (KNN), Approximate Nearest Neighbors (ANN), and Inverted File Index (IVF).

Accuracy at k=1:
KNN achieves an accuracy of 0.8781, which is the highest among the three methods, matching the performance of IVF, which also reaches 0.8781. On the other hand, ANN shows significantly lower accuracy, achieving only 0.5401. This suggests that while KNN and IVF are highly effective at identifying the correct image for a given text query, ANN sacrifices some accuracy for the sake of efficiency.

Average Comparisons at k=1:
In terms of computational efficiency, KNN requires the most comparisons, averaging 10.0 comparisons at k=1. ANN improves on this by reducing the average comparisons to 6.1, but still with a significant drop in accuracy. IVF performs the best in this regard, requiring only 3.0 comparisons on average while maintaining the same accuracy as KNN. This shows that IVF strikes an optimal balance between accuracy and computational efficiency, making it a preferable choice when both speed and performance are important.

Overall, IVF stands out as the most efficient and accurate method for text to image retrieval, offering the same accuracy as KNN but with fewer comparisons.

3 The Crypto Detective

Note: The dataset shall be in the same format is given in the link in the assignment document, inside a directory called q3_data

3.1 Correlation Matrix

The correlation matrix reveals some strong relationships between variables. For example:

- **Avg Value Received** is strongly correlated with **Min Value Received**.
- **Avg Value Received** also shows a strong correlation with **Avg Value Sent**.

However, most of the variables are not highly correlated. In some cases, negative correlations are observed, such as:

- **Avg Value Sent** and **Total Ether Balance** are negatively correlated, indicating that higher sent values tend to correspond with lower ether balances.

3.2 Violin Plots Analysis

1. **Transaction Frequency:**
    - The distributions of **Avg Minutes Between Sent Transactions** and **Avg Minutes Between Received Transactions** suggest that flagged accounts typically have shorter transaction time gaps. This indicates that flagged accounts tend to transact more frequently compared to non-flagged accounts.
2. **Transaction Values:**

- The distributions of **Min Value Received**, **Max Value Received**, and **Avg Value Received** show significant differences between flagged and non-flagged accounts. Flagged accounts tend to have more extreme values.
- **Min Value Sent** and **Avg Value Sent** also show higher variations in flagged accounts.

3. **Unique Addresses & Total Transactions:**

   - Flagged accounts tend to have a higher number of unique received addresses, which may indicate a pattern of interactions with many different senders.
   - **Total Transactions** (including contract creation) are significantly higher in flagged accounts.

4. **Total Ether Balance & Received Amounts:**

   - There is a noticeable difference in **Total Ether Balance** between flagged and non-flagged accounts. Flagged accounts often have extreme values.

3.3 Comparing my custom tree to scikit learn's default tree

3.3.1 accuracy comparison

Train Accuracy: Scikit-learn's tree achieves near-perfect training accuracy (0.9997) compared to the custom tree (0.8106).
Both implementations have similar validation accuracies (0.6565 for custom, 0.6428 for scikit-learn).
Scikit-learn's tree outperforms the custom tree on the test set (0.6224 vs. 0.4769).
This confirms the slightly better generalization of the scikit-learn mode

3.3.2 Computation time

Scikit-learn's tree is orders of magnitude faster (0.07 seconds) than the custom implementation (449 seconds). This is a crucial difference. Scikit-learn's implementation is highly optimized, leveraging efficient data structures(like numpy arrays) and algorithms (often written in C/C++). Custom implementations, especially in pure Python, are likely to be significantly slower.
Default parameters:Scikit learn uses gini impurity as the splitting criterion and has different min samples for different depths. It decides the max depth based on the min samples.

Custom Tree - Train Accuracy: 0.8105751186665194, Validation Accuracy: 0.6565121412803532, Test Accuracy: 0.47686329918756626

Custom Tree - Computation Time: 449.20496010780334 seconds
Scikit-learn Tree - Train Accuracy: 0.9996688376200463, Validation Accuracy: 0.6428256070640177, Test Accuracy: 0.622394913458142
Scikit-learn Tree - Computation Time: 0.0737297534942627 seconds

3.4 Hyper parameter tuning

Max Depth: 3, Min Samples: 5, Criterion: entropy, Validation Accuracy: 0.6044150110375276
Max Depth: 3, Min Samples: 5, Criterion: gini, Validation Accuracy: 0.6101545253863134
Max Depth: 3, Min Samples: 50, Criterion: entropy, Validation Accuracy: 0.6044150110375276
Max Depth: 3, Min Samples: 50, Criterion: gini, Validation Accuracy: 0.6101545253863134
Max Depth: 5, Min Samples: 5, Criterion: entropy, Validation Accuracy: 0.6172185430463576
Max Depth: 5, Min Samples: 5, Criterion: gini, Validation Accuracy: 0.6565121412803532
Max Depth: 5, Min Samples: 50, Criterion: entropy, Validation Accuracy: 0.6172185430463576
Max Depth: 5, Min Samples: 50, Criterion: gini, Validation Accuracy: 0.6565121412803532
Max Depth: 8, Min Samples: 5, Criterion: entropy, Validation Accuracy: 0.5969094922737307
Max Depth: 8, Min Samples: 5, Criterion: gini, Validation Accuracy: 0.6163355408388521
Max Depth: 8, Min Samples: 50, Criterion: entropy, Validation Accuracy: 0.5969094922737307
Max Depth: 8, Min Samples: 50, Criterion: gini, Validation Accuracy: 0.6141280353200883

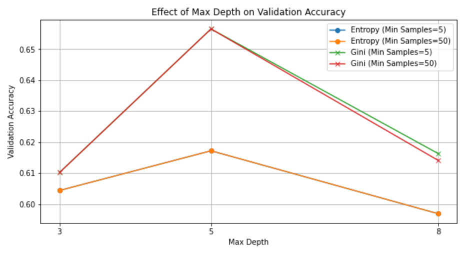

Figure 7: Enter Caption

3.4.1 Explanation:Bias-Variance Tradeoff

The goal of hyperparameter tuning is to find the sweet spot in the bias-variance tradeoff. A shallow tree (low max depth) has high bias (underfitting) – it doesn't capture the complexity of the data. A deep tree (high max depth) has high variance (overfitting) – it memorizes the training data too well and doesn't generalize well.

4 K-Means

Note: The video frames shall be in a directory called *frames* and there will another frame_0000.jpg

4.1 Affect of Compactness factor and number of cluster

Increasing the number of clusters leads to more number of superpixels and fine grained segmentation of image along the boundaries of it's components
Increasing the compactness factor has the effect of giving more weight to spatial proximity than color-similarity. Reducing it makes the components take up weird shapes due to color similarity.

LAB space seems to be slightly better suited than RGB space when performing superpixelation.
LAB space has a separate channel for controlling luminosity(L) and colour in RED-GREEN spectrum is controlled by A and Yellow-BLUE spectrum controlled by B. Due to this, the magnitude of change in colour in LAB space roughly equally corresponds to the percieved change in colour by humans. This is the reason behind LAB space's better performance compared to RGB space.

4.2 Optimization

I perform complete clustering starting from random centres for the first frame. For the successive frames, the previous frame's clusters give a good heuristic on how the clusters will look. So I initialize the cluster centres to previous frames final cluster centres coordinates and update the centres a few times.
In the sub-optimal clustering I run 10 iterations per frame. There are 11 frames, so it takes 110 iterations of updating clusters. In the optimized method, I do 10 iterations for the first frame and 2 iterations for the successive frames. So it takes 30 iterations(nearly 4x speed).
Average iterations per frame(sub optimal): 10
Average iterations per frame(optimal): 2.72