Statistical Methods in AI (Spring 2025)

Assignment-1
Deadline: 10th February, 2025

1 General Instructions

- Your assignment must be implemented in Python.
- Submit your assignment as a folder with a Jupyter Notebook files (.ipynb)
- The Deadline will not be extended.
- Submit a single report for all sections of the assignment.

2 Predicting Food Delivery Time (30 Marks)

This part of the assignment is designed to help you understand the fundamental concepts of linear regression and gradient descent while working with a real-world dataset on food delivery times. You are given a dataset (CSV file Link) that captures various factors influencing food delivery times, such as distance, weather, traffic conditions, time of day, vehicle type, preparation time, and courier experience. You are required to work on the following subtasks:

2.1 Exploratory Data Analysis (EDA) (3 marks)

- Load the dataset, handle any missing/Nan values and encode the categorical features using an appropriate method.
- Plot histograms, boxplots, violin plots, etc. for the features (choose plots based on type of feature). State your observations/insights.
- Split the dataset into train, test and validation sets (70:15:15 split) and scale the features.

2.2 Linear Regression with Gradient Descent (15 marks)

- Implement Batch, Mini-Batch and Stochastic gradient descent from scratch.
- Initialize the weights and bias to zero/random values (compare both in terms of performance, use either for future tasks) and train the model using the Mean Squared Error (MSE) loss function.
- Decide on the number of iterations (for a learning rate, say 0.05) by observing the convergence of the loss function during training, ensuring that the loss decreases steadily without fluctuations or divergence and stops improving significantly over iterations.
- Plot the loss function (MSE) vs. the number of iterations for each gradient descent method. Also plot the performance on validation set as the model trains (say measure once after every 10 iterations).
- Report and compare the final test set MSE and R-2 score for each method.

2.3 Regularization (8 marks)

Ridge and Lasso Regression are used to regularize linear models to avoid overfitting and improve predictive performance. Both methods add a penalty term to the model's cost function to constrain the coefficients, but they differ in how they apply this penalty. Ridge penalty is proportional to the square of the magnitude of the coefficients (weights), while Lasso adds a penalty which is based on the absolute values of the coefficients. Perform the the steps given below for both Lasso and Ridge (separately).

- Choose an appropriate regularization parameter (λ), say 0.5.
- Train the linear model using stochastic gradient descent. Report the performance metrics for test set (like previous task).
- Now, vary λ (say, 10 values between 0 and 1) and train the model. Plot the test MSE vs. λ and analyze the impact of regularization strength on model performance and overfitting.

2.4 Report (4 marks)

In addition to the analysis required for the above tasks, report the following (single PDF):

- Plot the line of best fit with the true values for train, test and vaidation set (for each unique model, post training).
- Explain the differences between the three types of gradient descent algorithms, along with their advantages and disadvantages.
- Which gradient descent method converged the fastest?
- How did Lasso and Ridge regularization influence the model? What is the optimal lambda (amongst the ones you have chosen) for Lasso and Ridge based on test performance?
- How does scaling of features affect model performance?
- Using a Barplot, describe how the trained model weights (1 per feature) vary in case of best performing model for Lasso and Ridge, compared to non-regularized model (best).
- Based on values of feature weights, analyze the effect of each feature on delivery time. Are there any features that have 0 (or almost 0) effect?

Note: You will not be evaluated based on how good your performance metrics are i.e. how close they are to the very best model possible (for some set of hyperparameters). So, do not spend much time finding the very best learning rate, number of iterations, etc.

3 KNN and ANN

(30 Marks)
CLIP (Contrastive Language–Image Pretraining) is a powerful neural network architecture developed by OpenAI that learns to associate text and images through a shared embedding space. Text descriptions and their corresponding images have embeddings close to each other in this space. In this task you have been given CLIP embeddings for the CIFAR-10 dataset. The train embeddings, test embeddings and corresponsing train labels and test labels can be found here. In addition, you will find text embeddings as well. The embeddings have been derived as follows: 1) We take each of the classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck in that order and encode them with the CLIPs text encoder as: "This is a {class}" to get the text embeddings. 2) We use the vision encoder of clip to get the train and test embeddings. Using these embeddings you will be asked to do various tasks related to KNN and Approximate Nearest Neighbour(ANN) search

3.1 KNN

(8 Marks)
Implement KNN from scratch, which takes a set of embeddings as input for the database. Given a query, the value of k and a distance metric, it returns the indices of the k nearest neighbours from the database.

Task 1: Classification

- Classify each image in the test set using the labels of the k nearest neighbours from the train set. Report the accuracy for 3 different values of k(1, 5, 10) and cosine and eucledian distance metrics (2 Marks).
- Instead of using the train set to predict the labels, now use the text embeddings to predict the labels. Using k = 1, get the text embedding closest to each image and predict the accuracy (2 Marks)

Task2: Retrieval

- Text to Image Retrieval: For each text embedding retrieve the 100 nearest embeddings from the train set and report the 3 metrics: Mean reciprocal rank, precision@100 and hit rate. (2 Marks)
- Image to Image Retrieval: For each test embedding retrieve the 100 nearest embeddings from train set and report the metrics mentioned above. (2 Marks)

3.2 Locally Sensitive Hashing

(10 Marks)
FAISS (Facebook AI Similarity Search) is a library developed by Meta AI for efficient similarity search and clustering of dense vectors. It is widely used for large-scale nearest-neighbor search, especially in applications involving high-dimensional data like image or text embeddings. FAISS offers different types of indexes, depending on the use case and dataset size. We have already seen two of those IndexFlatL2 and IndexFlatIP in previous section. Here we will implement LSH. For references one can look at FAISS github and LSH detailed. Note: You have to implement LSH and not use FAISS directly.

Task

- Implement LSH following the article linked above using random hyperplanes.
- Plot histograms showing frequency of samples in each bucket. Do this for different number of hyperplanes. What problems do you notice here?
- Perform image to image retrieval for your choice of k and report metrics mentioned in the previous section.
- How do these metrics change, on changing the number of hyperplanes, mention possible reasons for these.

Note: You can make reasonable assumptions. Eg:
1) If the nearest bucket has greater number of samples than k, then choose randomly or rank with distance and choose. The choice might change certain analysis performed later.
2) In the opposite case, choose the next bucket and so on until length becomes k or return lesser number of nearest neighbours.

3.3 IVF

(8 Marks)
IVF stands for Inverted File index. Similar to LSH, it aims to reduce the number of comparisons. You can refer to FAISS github to see how IVF works.

Task,

- Implement IVF from scratch. Use Kmeans algorithm for clustering. Since the dataset has 10 classes one can assign clusters as 10 but you are free to experiment.
- Perform image to image retrieval again and report the metrics for different value of nprobe.
- Plot the number of points in each cluster. Plot the average number of comparisons done for each query vs nprobe.

3.4 Analysis

(4 Marks)

- Compare the three methods based on performance on the common task. Plot necessary graphs.
- Compare the three methods based on average number of comparisons done per query. Plot the relevant graphs.

4 The Crypto Detective (30)

In the world of digital finance, **Mr. Richard** has a critical mission – and he needs your help. As a data detective at CryptoGuard Solutions, he's investigating potential fraud in cryptocurrency transactions.

Richard has an anonymous dataset of **Crypto X** filled with transaction details that might hide secrets about financial misconduct. But he can't crack this puzzle alone. He's looking for a skilled data scientist – maybe you – to help him.

His challenge to you: Can you dive into this dataset and help him separate legitimate transactions from suspicious ones? He believes the right analysis could protect countless investors from potential financial risks.
For your reference Richard has provided you with some resources: Link

Dataset Explanation

dataset link - SMAI'25-Detective

S.No	Column Name	Meaning
1	Address	Unique identifier of wallet, type: *OBJECT*
2	FLAG	Transaction safety indicator (0 = Safe, 1 = Fraud)
3	Avg min between sent tnx	Time gap between sending money
4	Avg min between received tnx	Time gap between receiving money
5	Time Diff between first and last (Mins)	Total active transaction time
6	Unique Received From Addresses	Number of different sender addresses
7	min value received	Smallest amount of money received
8	max value received	Largest amount of money received
9	avg val received	Average amount of money received
10	min val sent	Smallest amount of money sent
11	avg val sent	Average amount of money sent
12	total transactions	Total number of transactions
13	total crypto received	Total money received
14	total crypto balance	Current wallet money balance

Table 1: Description of Cryptocurrency Transaction Data

4.1 Preprocessing and Exploratory Data Analysis (5)

- Load the dataset, clean the data by removing unnecessary features, handle any missing/Nan values and scale the features with large ranges to ensure consistency.

- Plot the correlation matrix for the 13 features (exclude Address), and provide your observations.
- Plot the variation of other columns with respect to the *FLAG* column using a violin plot and write your observations about the graphs.

4.2 Fraud Detection Model (15)

- **Richard** has suggested you develop a decision tree algorithm from scratch using Classes in Python for solving this task. Incorporate *max_depth* and *min_samples* parameters to control tree growth, ensuring robust and customizable tree construction. Implement Eentropy and Gini Impurity as splitting criteria.
- Compare your custom decision tree implementation with `scikit-learn`'s built-in implementation by evaluating accuracies on train, validation, and test sets. Analyze the performance differences, computation time, and potential sources of variation, such as implementation details, default hyperparameters, and underlying algorithm, between your scratch implementation and the library version.

4.3 Hyperparameter Tuning (5)

Giving your best possible model is always a need of the CryptoGuard Solutions, so:

- Explore key decision tree hyperparameters like maximum tree depth and minimum samples for split, and compare entropy and Gini impurity metrics.
- Utilize the validation set for hyperparameter tuning, creating accuracy plots to visualize performance across different model configurations.
- Why do you think you obtained these results? Provide a possible explanation.

4.4 Model Visualization (5)

- **Feature Importance Visualization:** Develop bar plots or color-coded charts to graphically represent feature contributions, highlighting the most influential predictors in the model.
- **Tree Visualization:** Create a graphical decision tree representation, revealing the model's decision-making structure.

5 K-Means

(30 Marks)
K-Means is a popular unsupervised machine learning algorithm used for clustering data into k distinct groups (or clusters) based on similarity. SLIC(Simple Linear Iterative Clustering) is a fast and efficient algorithm used for superpixel segmentation in images. It groups pixels into small, uniform regions (superpixels) that have similar color and spatial proximity. It is a basic extension of the kmeans algorithm. More about slic can be found here. All the necessary files and scripts for this section can be found here

5.1 SLIC

(15 Marks)
Task

- Refer to the article and implement SLIC from scratch.
- Try to segment the image frame_0000.jpg. Alternatively, choose any image and try segmentation/superpixalation for it using SLIC.
- Change the two hyperparameters: number of clusters and compactness and report the difference they bring.

- Display the image after every iteration to qualitatively see the change that happens in every iteration
- As you can see the distance metric used is in Lab Space and not RGB space. Try out the RGB space and report any difference noticed. What might be the reason for it.

Note:

- Assign centers randomly at the start on any point of image.
- Convergence condition is not necessary. Just play around with number of iterations and find the one that qualitatively seems the best.
- openCV is the library used for image processing. Some functions that one might need are cv2.imread. The image might load as BGR instead of RGB. To handle these, one can use cv2.cvtColor(image, cv2.COLOR_BGR2RGB). After the segmentation is done, and labels are assigned to each pixel, the segmented image can be generated by label2rgb from scikit image package. Feel free to reach out for any more doubts about handling images.

5.2 Video Segmentation with SLIC

(8 Marks)
Task

- The video titled segment.mp4 in the google drive has been cropped to input.mp4. Frames segmented out of the video, have been added to the frames folder. The code to generate video from frames and extract frames from video can be found in preprocess.ipynb.
- Using the Slic algorithm written above, do a frame wise segmentation of the video. Generate back the video from the frames using the script.

5.3 Optimizing the video segmentation

(7 Marks)
Task

- As you can see, it takes 10x the time to segment the video fram-wise. But, since this is a video, there is also some temporal information associated.
- While one of the frames might take the same number of iterations, can you reduce the number of iterations needed for subsequent frames using simple changes.
- Similar to previous task generate a video from the segments and mention the method used for the optimization.
- Compare the average number of iterations you have used per frame in the two cases.