NORTHEASTERN UNIVERSITY
Department of Electrical and Computer Engineering

EECE 5644 Machine Learning and Pattern Recognition

in

Homework Assignment - 2

Omkar Rajendra Gaikwad
NUID: 002711498
March 13 2023
Github Link for Output and python files

Question 1

The probability density function (pdf) for a 2-dimensional real-valued random vector X is as follows:

$$p(x) = P(L = 0)p(x|L = 0) + P(L = 1)p(x|L = 1).$$

Here L is the true class label that indicates which class-label-conditioned pdf generates the data. The class priors are $P(L = 0) = 0.6$ and $P(L = 1) = 0.4$. The class-conditional pdfs are

$$p(x|L = 0) = w_{01}g(x|m_{01}, C_{01}) + w_{02}g(x|m_{02}, C_{02})$$

and

$$p(x|L = 1) = w_{11}g(x|m_{11}, C_{11}) + w_{12}g(x|m_{12}, C_{12}),$$

where $g(x|m, C)$ is a multivariate Gaussian probability density function with mean vector m and covariance matrix C. The parameters of the class-conditional Gaussian pdfs are: $w_{i1} = w_{i2} = 1/2$ for $i \in 1, 2$, and

$$m_{01} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, m_{02} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, m_{11} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, m_{12} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$C_{ij} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

for all i, j pairs.

For numerical results requested below, generate the following independent datasets each consisting of iid samples from the specified data distribution, and in each dataset make sure to include the true class label for each sample:

- D_{20} train consists of 20 samples and their labels for training;
- D_{200} train consists of 200 samples and their labels for training;
- D_{2000} train consists of 2000 samples and their labels for training;
- D_{10K} validate consists of 10000 samples and their labels for validation.

Part 1: (6%) Determine the theoretically optimal classifier that achieves the minimum probability of error using the knowledge of the true pdf. Specify the classifier mathematically and implement it. Then apply it to all samples in D_{10K} validate. From the decision results and true labels for this validation set, estimate and plot the ROC curve for a corresponding discriminant score for this classifier, and on the ROC curve indicate, with a special marker, the location of the min-$P(error)$ classifier. Also, report an estimate of the min-$P(error)$ achievable, based on counts of decision-true label pairs on D_{10K} validate. Optional: As supplementary visualization, generate a plot of the decision boundary of this classification rule overlaid on the validation dataset. This establishes an aspirational performance level on this data for the following approximations.

Part 2: (12%) (a) Using the maximum likelihood parameter estimation technique, train three separate logistic-linear-function-based approximations of class label posterior functions given

a sample. For each approximation, use one of the three training datasets D_{20} train, D_{200} train, D_{2000} train. When optimizing the parameters, specify the optimization problem as minimization of the negative-log-likelihood of the training dataset, and use your favorite numerical optimization approach, such as gradient descent or Matlab's fminsearch. Determine how to use these class-label-posterior approximations to classify a sample in order to approximate the minimum-$P(error)$ classification rule. Apply these three approximations of the class label posterior function on samples in D_{10K} validate, and estimate the probability of error that these three classification rules will attain (using counts of decisions on the validation set). Optional: As supplementary visualization, generate plots of the decision boundaries of these trained classifiers superimposed on their respective training datasets and the validation dataset.

(b) Repeat the process described in Part (2a) using a logistic-quadratic-function-based approximation of class label posterior functions given a sample.

Discussion: (2%) How does the performance of your classifiers trained in this part compare to each other considering differences in the number of training samples and function form? How do they compare to the theoretically optimal classifier from Part 1? Briefly discuss results and insights. Note 1: With x representing the input sample vector and w denoting the model parameter vector, logistic-linear-function refers to $h(x, w) = 1/(1 + e^{-w^T z(x)})$, where $z(x) = [1, x^T]^T$; and logistic-quadratic-function refers to $h(x, w) = 1/(1 + e^{-w^T z(x)})$, where $z(x) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]^T$.

0.0.1 Answer Part 1:

1. The given "variable.py" file imports necessary libraries including numpy, pandas, matplotlib, and scipy. It also sets the random seed for numpy to ensure reproducibility.
   The code defines a probability density function (pdf) as a dictionary with keys "prior", "gmm_w", "mo", and "Co", which represent the prior probabilities, Gaussian mixture model weights, means, and covariance matrices, respectively. It then initializes empty lists X_train, labels_train, and N_labels_train. The function generate_data(N, pdf) takes two arguments, N (an integer) and pdf (the probability density function dictionary defined earlier) and generates synthetic data based on the specified PDF. The function first generates N random numbers (u) between 0 and 1. It then computes thresholds based on the prior probabilities and GMM weights. Next, the function generates labels by comparing the randomly generated u to the prior probability of belonging to the first class. The function then generates data for each Gaussian component of the GMM, depending on which threshold each random number u falls within. Finally, the function returns the generated data (X) and labels (labels).
   Overall, this code generates synthetic data based on a specified probability density function and is useful for generating data for testing and experimentation purposes.

2. The given "sample_plot.py" file generates and visualizes training and validation datasets for a binary classification problem.

The code imports various libraries including numpy, pandas, matplotlib, scipy, and sklearn. It also imports two functions (pdf and generate_data) and three variables (X_train, labels_train, N_labels_train) from an external file named variable.py. The code then creates a fig object using plt.figure function with a size of 14x14. It divides the figure into four subplots using the fig.add_subplot function and sets the titles, x-axis and y-axis labels for each of the subplots.
The first three subplots correspond to the training datasets with 20, 200, and 2000 samples respectively. For each of these subplots, the code generates the corresponding dataset and their labels using the generate_data function and appends them to the X_train, labels_train, and N_labels_train lists. The scatter function from matplotlib is used to visualize the dataset points with different colors and markers for the two classes.
The fourth subplot corresponds to the validation dataset with 10000 samples. The validation dataset is also generated using the generate_data function and visualized using the scatter function.
Finally, the code sets the x and y limits of the subplots using plt.setp function, adjusts the layout of the subplots using plt.tight_layout, saves the figure using plt.savefig, and displays the figure using plt.show.
Figure 1. shows the scatter plot of the Samples for 20, 200, 2000 and 10000 samples for Validation dataset.

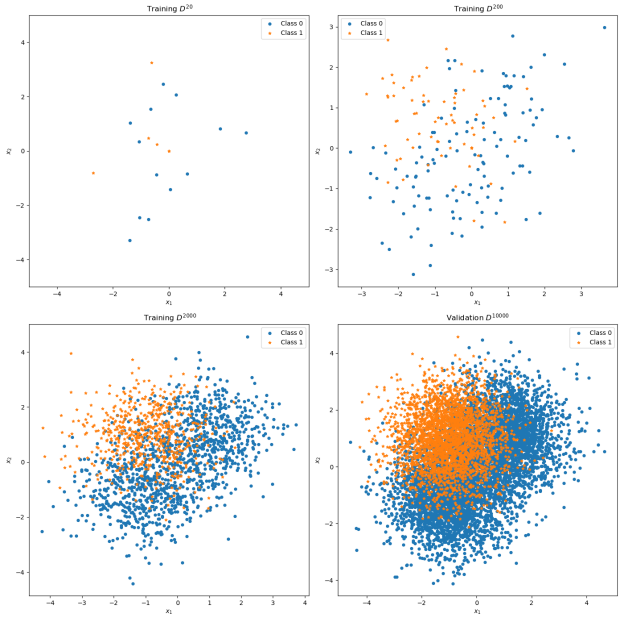

Figure 1: Scatter plot of 20, 200, 2000, and 10,000 samples for training and validation dataset

3. The given "evaluation_function.py" file provided is a script that performs binary classification using the ERM (Empirical Risk Minimization) algorithm and plots the resulting decision boundary. The main function used for classification is erm_discriminant,

which calculates the ERM score for a given input X and a probability density function (pdf) that models the two classes. The function returns the difference between the logarithm of the likelihoods of the two classes.
The ROC (Receiver Operating Characteristic) curve is then estimated using the estimate_roc function, which takes as input the ERM scores, the true labels, and the number of samples for each class. The ROC curve is represented as the plot of the true positive rate (TPR) against the false positive rate (FPR) for different thresholds.
The get_binary_classification_metrics function is used to compute several performance metrics for binary classification, such as true positive rate (TPR), false positive rate (FPR), true negative rate (TNR), and false negative rate (FNR).
Finally, the plot_erm function is used to plot the decision boundary obtained by the ERM algorithm using the contour function from the matplotlib library. The boundary is calculated by calling the prediction_score function, which evaluates the ERM score for a grid of points in the input space. The resulting scores are then reshaped into a two-dimensional array, which is used to generate the contour plot. The function also takes care of labeling the contours with their corresponding values.
In summary, the provided Python code performs binary classification using the ERM algorithm and provides useful visualizations of the resulting decision boundary and ROC curve.

4. To generate an ROC curve of an implementation of the Empirical Risk Minimization (ERM) algorithm for binary classification tasks we wrote specific code.
The algorithm tries to find the classifier that minimizes the empirical risk (error) based on the training data.
The code assumes that the training data is stored in the X_valid and labels_valid arrays, where X_valid contains the feature vectors of the training data and labels_valid contains their corresponding binary labels (0 or 1). It also assumes that the prior probabilities of the two classes are stored in the pdf dictionary under the keys prior[0] and prior[1].
The code first computes the discriminant scores for the validation set using the erm_discriminant(). The estimate_roc() function is then used to estimate the Receiver Operating Characteristic (ROC) curve of the classifier based on the validation set. The ROC curve is then plotted using matplotlib.
The code also computes the minimum empirical error probability and the corresponding gamma value using the ROC curve. The get_binary_classification_metrics() function is used to compute the false positive rate (FPR) and true positive rate (TPR) of the classifier based on the validation set. The minimum theoretical error probability and the corresponding gamma value are also computed based on the FPR, TPR, and the prior probabilities.
Finally, the code prints the minimum empirical and theoretical error probabilities and their corresponding gamma values.
Figure 2 shows the plot of the ROC curve.

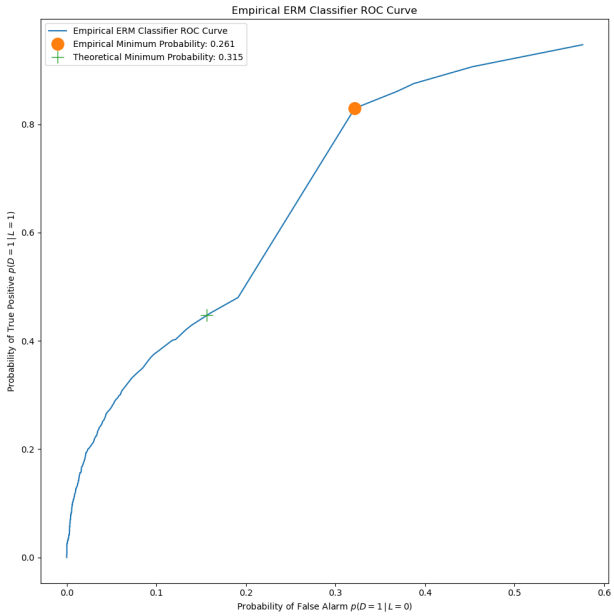

Figure 2: Normal ROC Curve Generated.

5. **Optional:** To plot the validation set with the decision boundary of the ERM classifier. The prediction_score() function takes in X_bound and Y_bound which are tuples of the lower and upper bounds for the x and y axes respectively. It also takes in pdf which is the dictionary containing the prior probabilities, mean vectors, and covariance matrices for each class. If a feature transformation phi is passed in, it applies the transformation to a meshgrid created from X_bound and Y_bound. It then calculates the discriminant function values for each point on the grid using the erm_discriminant() function and returns the meshgrid xx and yy, and the Z values which correspond to the discriminant function values for each point on the grid.
   The plot_erm() function takes in an axis object ax, a dataset X, and the pdf dictionary. It calculates the bounds for the x and y axes and creates a meshgrid using these bounds with 200 points in each dimension. It then calls the prediction_score() function to get the discriminant function values for each point on the grid. It calculates levels for the contour plot by taking equal steps between 0 and the minimum and maximum discriminant function values. It then plots the contour lines on the axis object ax with the specified levels and colors.
   Finally, the code plots the validation set with the decision boundary of the ERM classifier. It first extracts the coordinates for the true negatives, false positives, false negatives, and true positives from the validation set using the binary classification metrics calculated earlier. It then creates a new figure with an axis object and plots the validation set points with different markers for each class. It then calls plot_erm() to plot the decision boundary on top of the scatter plot. It sets the aspect ratio to be equal and adds a legend to the plot.
   Figure 3. Shows the Contour Plot.

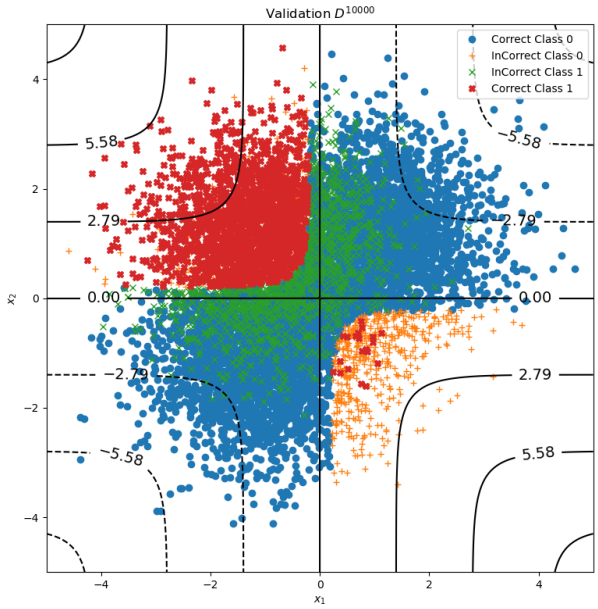

Figure 3: Decision Boundary Plot

0.0.2 Answer Part 2:

(a) To evaluate logistic regression classifier that is trained and evaluated on different sets of data. Below is a brief explanation of the code:

i. The code defines the value of Epi as 1e-7, which is used to avoid taking the logarithm of zero in the negative_log_likelihood function.

ii. The logistic_prediction function takes in the input data X and the weights w, and returns the predicted probabilities of the target variable using the logistic function.

iii. The negative_log_likelihood function takes in the true labels and predicted probabilities and calculates the negative log-likelihood loss.

iv. The Compute_param_logistic function takes in the input data and true labels, initializes the weights randomly, and uses the minimize function to find the optimal weights that minimize the negative log-likelihood loss.

v. The prediction_score_grid function takes in the x and y boundaries of the plot, a trained logistic regression model, and a data transformation function phi. It generates a grid of coordinates, applies the data transformation function phi if provided, and returns the predicted probabilities for the grid points.

vi. The logistic_classifier function takes in the plot axis, the input data X, the trained weights w, the true labels, the number of labels N_labels, and the data

transformation function phi. It generates the predicted labels, calculates the binary classification metrics, plots the data points, and adds the contour plot of the predicted probabilities.

vii. The code creates a PolynomialFeatures object phi of degree 1.

viii. The code creates a matplotlib figure and adds six subplots.

ix. For each of the three training sets, the code trains a logistic regression model using the Compute_param_logistic function and plots the data points and the decision boundary using the logistic_classifier function.

x. For each of the three validation sets, the code evaluates the trained logistic regression models and plots the data points and the decision boundary using the logistic_classifier function.

xi. The code generates a legend for the subplots and sets the x and y limits of the plots. The code shows the plots using the plt.show() function.

xii. For linear Logistic Model the equation used is:
$g(\mathbf{w}^\intercal \phi(\mathbf{x})) = \frac{1}{1+e^{-\mathbf{w}^\intercal \phi(\mathbf{x})}}$ where $\phi(\mathbf{x}) = \left[1, \mathbf{x}\right]^\intercal$ represents the augmented input vector $\tilde{\mathbf{x}}$.

xiii. For Quadratic Logistic Model the equation used is:
$g(\mathbf{w}^\intercal \phi(\mathbf{x})) = 1/(1 + e^{-\mathbf{w}^\intercal \phi(\mathbf{x})})$ with $\phi(\mathbf{x}) = \left[1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\right]^\intercal$.

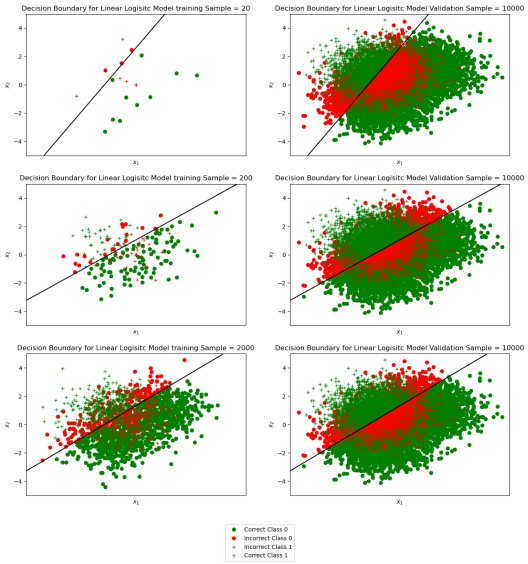

Figure 4: Decision Boundary for Logistic Linear Model

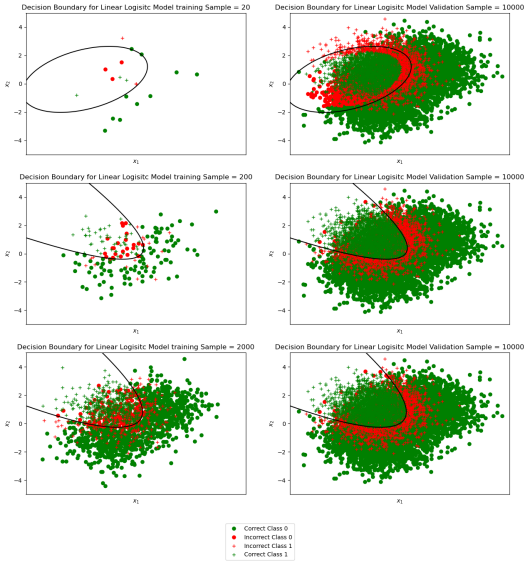

Figure 5: Decision Boundary for Logistic Quadratic Model

xiv. Figure 4. displays the linear Model Contours, and Figure 5. shows the Quadratic model.

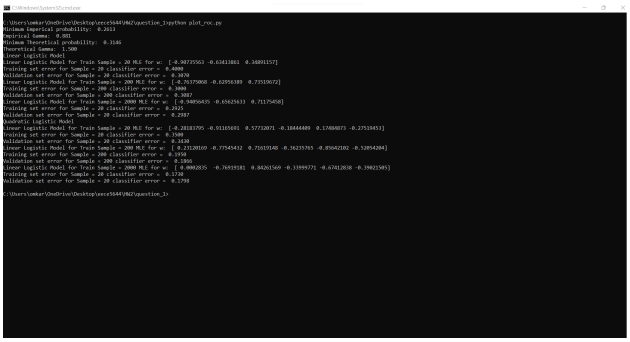

Figure 6: Optimization of the loss for a logistic model applied to different training subsets

(b) The given results in Figure 6. show the performance of Linear Logistic and Quadratic Logistic Models on the training and validation sets for three different training sample sizes: 20, 200, and 2000. The Maximum Likelihood Estimates (MLE) for the weight vector $\mathbf{w}$ for each model and sample size are also provided. For the Linear Logistic Model, we see that as the training sample size increases, the error rates on both the training and validation sets decrease. However, even for the largest sample size (2000), the error rates are still relatively high (training error of 0.2925 and validation error of 0.2987). This suggests that a linear decision boundary is not very effective at separating the two classes in the given dataset. On the other hand, for the Quadratic Logistic Model, we see that the error rates

on both the training and validation sets decrease significantly as the training sample size increases. For the largest sample size (2000), the error rates are quite low (training error of 0.1730 and validation error of 0.1798). This suggests that a quadratic decision boundary is a much better fit for the dataset and can separate the two classes more effectively than a linear decision boundary.

Furthermore, we can compare the error rates of the Linear Logistic Model and Quadratic Logistic Model directly. For the smallest training sample size (20), the Quadratic Model performs slightly worse than the Linear Model on the validation set (error rates of 0.3430 and 0.3070, respectively). However, as the sample size increases, the Quadratic Model outperforms the Linear Model by a significant margin on both the training and validation sets. For example, for the largest training sample size (2000), the Quadratic Model has a validation error rate of 0.1798 compared to the Linear Model's validation error rate of 0.2987.

In summary, the results suggest that a Quadratic Logistic Model is a better fit for the given dataset than a Linear Logistic Model, and as the sample size increases, the Quadratic Model performs significantly better than the Linear Model on both the training and validation sets.

1 Question 2:

Assume that scalar-real y and two-dimensional real vector $\boldsymbol{x}$ are related to each other according to $y = c(\boldsymbol{x}, \boldsymbol{w}) + v$, where $c(., \boldsymbol{w})$ is a cubic polynomial in $\boldsymbol{x}$ with coefficients $\boldsymbol{w}$ and v is a random Gaussian random scalar with mean zero and σ^2-variance. Given a dataset $D = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_N}, y_N)$ with N samples of $(\boldsymbol{x}, y)$ pairs, with the assumption that these samples are independent and identically distributed according to the model, derive two estimators for $\boldsymbol{w}$ using maximum-likelihood (ML) and maximum-a-posteriori (MAP) parameter estimation approaches as a function of these data samples. For the MAP estimator, assume that $\boldsymbol{w}$ has a zero-mean Gaussian prior with covariance matrix $\gamma \boldsymbol{I}$. Having derived the estimator expressions, implement them in code and apply to the dataset generated by the attached Matlab script. Using the training dataset, obtain the ML estimator and the MAP estimator for a variety of γ values ranging from 10^{-m} to 10^{n}. Evaluate each trained model by calculating the average-squared error between the y values in the validation samples and model estimates of these using $c(., \boldsymbol{w}_{\text{trained}})$. How does your MAP-trained model perform on the validation set as γ is varied? How is the MAP estimate related to the ML estimate? Describe your experiments, visualize and quantify your analyses (e.g. average squared error on validation dataset as a function of hyperparameter γ) with data from these experiments.

Note: Point split will be 20% for ML and 20% for MAP estimator results and discussion.

1.0.1 Answer 2

(a) To solve the question, We generated data for Question 2 here is the summary of the code:

i. generateDataFromGMM(N, gmm_pdf) is a function that generates a dataset of size N from a Gaussian Mixture Model (GMM) specified by the dictionary gmm_pdf. The GMM is defined by the prior probabilities of the classes, the mean vectors of the Gaussians, and their covariance matrices. The function returns a tuple of two arrays: the first array contains the 2D coordinates of the data points, and the second array contains the class labels.

ii. generate_data(N) is a function that generates a dataset of size N with three classes. The dataset is created by calling generateDataFromGMM() with a pre-defined GMM. The function returns the same tuple as generateDataFromGMM().

iii. prediction_score(X_bound, Y_bound, pdf, prediction_function, phi=None, num_cord=100) is a function that evaluates the performance of a prediction function on a grid of points. The grid is defined by the x and y coordinate bounds X_bound and Y_bound, and the number of points in each direction num_cord. The prediction function takes two arguments: a 2D array of points and a dictionary pdf that defines the GMM for the problem. If phi is not None, the points are transformed using a polynomial feature expansion. The function returns three arrays: the x and y coordinates of the points in the grid, and the predicted class labels for each point.

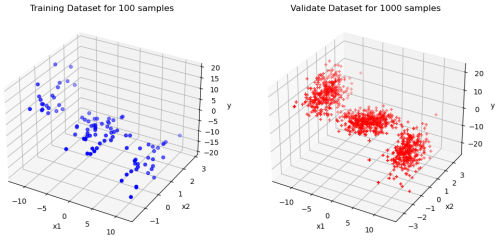

Figure 7: Scatter Plot for training and Validation data

iv. Figure 7 shows the scatter plot for the data.

(b) To find the MLE estimator and MAP estimator and Average Squared- Error here is the Summary of the code:

i. mle_estimator(X,y): This function takes in a matrix X and a vector y, and computes the maximum likelihood estimator for linear regression coefficients using the formula inv(X.T.dot(X)).dot(X.T).dot(y), where inv() represents matrix inverse and dot() represents matrix multiplication. It returns the computed values.

ii. map_estimator(X, y, gamma): This function takes in a matrix X, a vector y, and a regularization parameter gamma, and computes the maximum a posteriori (MAP) estimator for linear regression coefficients using the formula inv(X.T.dot(X) + (1 / gamma)*np.eye(X.shape[1])).dot(X.T).dot(y), where inv() represents matrix inverse, dot() represents matrix multiplication, and np.eye() creates an identity matrix. It returns the computed values.

iii. ase(y_pred, y_true): This function takes in two vectors y_pred and y_true, representing the predicted and true values of the target variable, respectively. It computes the average squared error (ASE) between the predicted and true values using the formula mean((y_pred - y_true) ** 2), where mean() computes the average of the given vector. It returns the computed value.

(c) To perform polynomial regression on a dataset and find MLE and MAP estimator following is the summary code.

i. Dataset generated by the function hw2q2(). The dataset is split into a training set with 100 samples and a validation set with 1000 samples.

ii. First, the required libraries are imported. These include numpy, pandas, matplotlib, and sklearn. Additionally, custom functions map_estimator(), mle_estimator(), ase(), prediction_score(), and hw2q2() are imported from the corresponding modules.

iii. The hw2q2() function generates a training set and a validation set of 100 and 1000 samples, respectively. The training set and validation set are plotted using 3D scatter plots.

iv. The PolynomialFeatures() function is used to transform the input features into higher degree polynomials. The degree of the polynomial is set to 3. The transformed features are used to train a maximum likelihood estimator (MLE) using the mle_estimator() function. The MLE estimates the coefficients of the polynomial regression model.

v. The MLE estimator is used to predict the output variable for the validation set, and the average squared error (ASE) between the predicted values and the actual values is computed using the ase() function. The ASE for the MLE estimator is printed.

vi. A surface plot is created using the predicted values and the original validation set, and this plot is saved as a PNG file.

vii. Next, a maximum a posteriori (MAP) estimator is trained using the map_estimator() function with a range of hyperparameters gamma. The hyperparameter that yields the lowest ASE on the validation set is chosen as the best hyperparameter for the MAP estimator. The ASE values for different hyperparameters are plotted against the corresponding hyperparameters using a log scale on the x-axis.

viii. Overall, the code performs polynomial regression on a dataset and compares the performance of the MLE and MAP estimators.

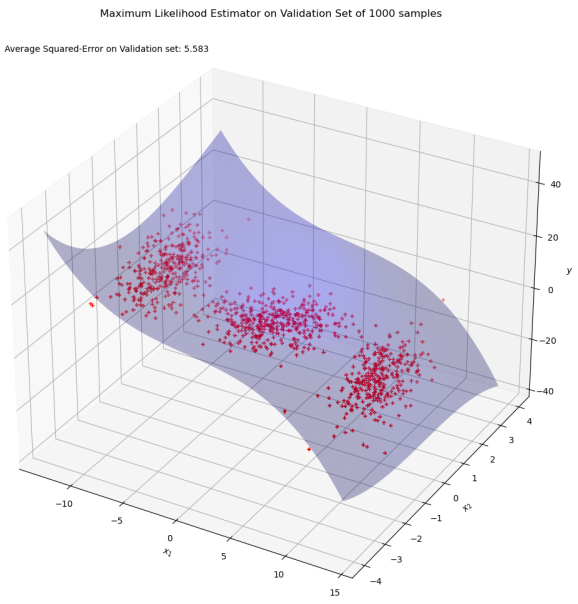

Figure 8: MLE Estimator

ix. Figure 8 and Figure 9 shows the MLE and MAP Estimator.

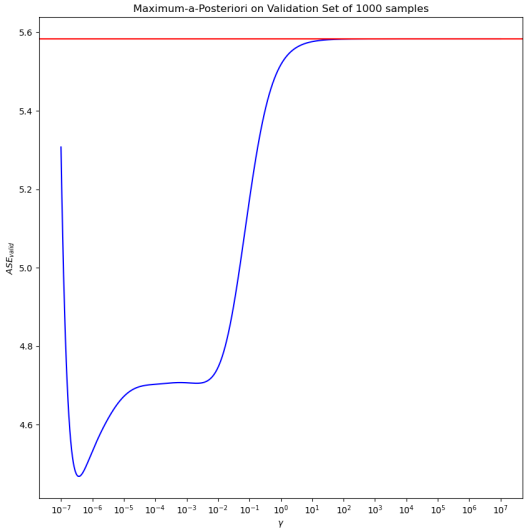

Figure 9: MAP Estimator

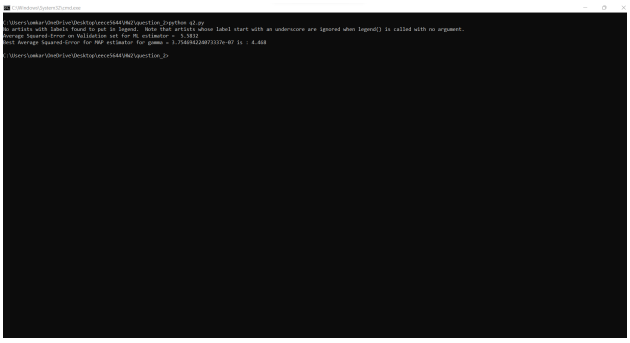

Figure 10: Output

(d) The ML estimator and MAP estimator are two different methods used to estimate the parameters of a model given some training data.
The ML estimator uses the maximum likelihood principle to find the values of the parameters that maximize the likelihood of observing the given data. In other words, it tries to find the parameters that make the observed data the most likely under the assumed model. In this code, the ML estimator is used to find the parameters of a linear regression model.
The MAP estimator, on the other hand, is a Bayesian approach that uses prior knowledge about the parameters to find the posterior distribution of the parameters given the data. It tries to find the values of the parameters that maximize the posterior probability. In this code, the MAP estimator is used with a regularization term (gamma) to find the parameters of a regularized linear regression model.
To compare the performance of the two estimators, the Average Squared-Error

(ASE) is calculated on a validation set. The ASE is a measure of the quality of the model's predictions, and it measures the average of the squared differences between the predicted values and the true values on the validation set.
The result of the ML estimator is an ASE of 5.5832, which indicates that the ML estimator is not performing well on the validation set. The result of the MAP estimator with the best gamma value (3.754694224073337e-07) is an ASE of 4.468, which indicates that the MAP estimator is performing better than the ML estimator on the validation set.
Overall, the MAP estimator is expected to perform better than the ML estimator when the data is noisy or when the number of features is large, because it can incorporate prior knowledge to avoid overfitting. However, it may require tuning of the regularization parameter (gamma) to achieve the best performance.
Figure 10. shows the Output for MLE and MAP Estimator.

2 Question 3

Question 3 (20%)

A vehicle at true position $[x_T, y_T]^{\mathrm{T}}$ in 2-dimensional space is to be localized using distance (range) measurements to K reference (landmark) coordinates $\{[x_1, y_1]^{\mathrm{T}}, \ldots, [x_i, y_i]^{\mathrm{T}}, \ldots, [x_K, y_K]^{\mathrm{T}}\}$. These range measurements are $r_i = d_{Ti} + n_i$ for $i \in \{1, \ldots, K\}$, where $d_{Ti} = \|[x_T, y_T]^{\mathrm{T}} - [x_i, y_i]^{\mathrm{T}}\|$ is the true distance between the vehicle and the i^{th} reference point, and n_i is a zero mean Gaussian distributed measurement noise with known variance σ_i^2. The noise in each measurement is independent from the others.

Assume that we have the following prior knowledge regarding the position of the vehicle:

$$p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = (2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \tag{1}$$

where $[x, y]^{\mathrm{T}}$ indicates a candidate position under consideration.

Express the optimization problem that needs to be solved to determine the MAP estimate of the vehicle position. Simplify the objective function so that the exponentials and additive/multiplicative terms that do not impact the determination of the MAP estimate $[x_{MAP}, y_{MAP}]^{\mathrm{T}}$ are removed appropriately from the objective function for computational savings when evaluating the objective.

Implement the following as computer code: Set the true vehicle location to be inside the circle with unit radious centered at the origin. For each $K \in \{1, 2, 3, 4\}$ repeat the following.

Place evenly spaced K landmarks on a circle with unit radius centered at the origin. Set measurement noise standard deviation to 0.3 for all range measurements. Generate K range measure-

ments according to the model specified above (if a range measurement turns out to be negative, reject it and resample; all range measurements need to be nonnegative).

Plot the equilevel contours of the MAP estimation objective for the range of horizontal and vertical coordinates from -2 to 2; superimpose the true location of the vehicle on these equilevel contours (e.g. use a + mark), as well as the landmark locations (e.g. use a *o* mark for each one).

Provide plots of the MAP objective function contours for each value of K. When preparing your final contour plots for different K values, make sure to plot contours at the same function value across each of the different contour plots for easy visual comparison of the MAP objective landscapes. *Suggestion:* For values of σ_x and σ_y, you could use values around 0.25 and perhaps make them equal to each other. Note that your choice of these indicates how confident the prior is about the origin as the location.

Supplement your plots with a brief description of how your code works. Comment on the behavior of the MAP estimate of position (visually assessed from the contour plots; roughly center of the innermost contour) relative to the true position. Does the MAP estimate get closer to the true position as K increases? Doe is get more certain? Explain how your contours justify your conclusions.

Note: The additive Gaussian distributed noise used in this question is likely not appropriate for a proper distance sensor, since it could lead to negative measurements. However, in this question, we will ignore this issue and proceed with this noise model for illustration. In practice, a multiplicative log-normal distributed noise may be more appropriate than an additive normal distributed noise depending on the measurement mechanism.

2.1 Answer 3

Part 1: Express the optimization problem that needs to be solved to determine the MAP estimate of the vehicle position. Let the true position of the vehicle be denoted by $x_0 = [x_0, y_0]^T$, and the position of the ith reference point be denoted by $p_i = [p_{ix}, p_{iy}]^T$. Let the candidate vehicle position under consideration be denoted by $x = [x, y]^T$. Assuming that the range measurements are independent and Gaussian with mean $||p_i - x||$ and standard deviation σ_r, the likelihood function for the range measurements can be written as:

$$p(z_1, \ldots, z_K | x_0) = \prod_{i=1}^{K} \mathcal{N}(z_i; ||p_i - x||, \sigma_r^2)$$

where z_i is the ith range measurement. Assuming that the prior on the vehicle position is a Gaussian with mean x_0 and covariance matrix R, the prior distribution can be written as:

$$p(x_0) = \mathcal{N}(x_0; \mu, R)$$

where μ is the mean of the prior distribution. The MAP estimate of the vehicle position can be obtained by maximizing the posterior distribution:

$$p(x_0 | z_1, \ldots, z_K) = \frac{p(z_1, \ldots, z_K | x_0) p(x_0)}{p(z_1, \ldots, z_K)}$$

Since the denominator is a normalization constant, it can be ignored for the purpose of optimization. Thus, the optimization problem can be written as:

$$\max_{x_0} \log(p(z_1, \ldots, z_K|x_0)p(x_0))$$

Expanding the squared terms and simplifying the objective function, we get:

$$\max_{x_0} -\sum_{i=1}^{K} \frac{(z_i - ||p_i - x||)^2}{2\sigma_r^2} - \frac{(x_0 - \mu)^T R^{-1}(x_0 - \mu)}{2}$$

Therefore, the optimization problem can be solved by minimizing the negative of the above objective function w.r.t. x_0.

Part 2: The Python code is implementing a simulation of a range-based localization problem. It defines a scenario where a vehicle is located at a true position, and several landmarks are placed on a circle around the vehicle. The objective is to estimate the position of the vehicle based on range measurements from the landmarks, where the range measurement is affected by Gaussian noise. The code is structured as follows:
First, the required libraries, NumPy and Matplotlib, are imported.
The true position of the vehicle is defined as a 2D NumPy array.
The noiserange variable is defined, which represents the standard deviation of the Gaussian noise affecting the range measurements.
The range_model function is defined, which takes a point as an input and returns a range measurement value with Gaussian noise added to it. This function uses the true_vehicle_position and noiserange variables to generate the range measurement.
The objective_function function is defined, which takes a point, an array of landmarks, and an array of ranges as inputs, and returns the value of the objective function for these inputs. The objective function is a sum of two terms: a prior term that favors points close to the origin (0,0), and an error term that measures the difference between the measured ranges and the ranges calculated from the distances between the landmarks and the point. The function returns the sum of these two terms.
The sample_range function is defined, which takes a landmark as input and generates a range measurement for that landmark using the range_model function. This function uses a rejection sampling technique to ensure that the range measurement is non-negative.
The plot_contour function is defined, which takes a value of K as input and generates a plot of the objective function contours for a scenario with K landmarks. The function first places the landmarks on a circle using the np.linspace function. It then generates range measurements for each landmark using the sample_range function.
The function defines a range for the x and y coordinates and creates a meshgrid

of points using np.meshgrid. For each point in the meshgrid, it calculates the objective function using the objective_function function and stores the value in a 2D NumPy array.
Finally, the function generates a contour plot of the objective function using plt.contour, where the levels of the contours are defined using np.logspace. It also plots the true position of the vehicle and the positions of the landmarks using plt.plot, and saves the plot as a PNG file.
The for loop at the end calls the plot_contour function for values of K from 1 to 4, generating four different plots of the objective function contours with different numbers of landmarks.

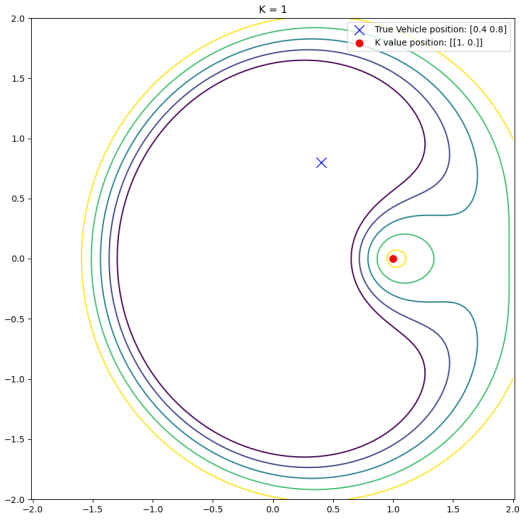

Figure 11: K=1

Figure 11 to 14. are the plots for K values.

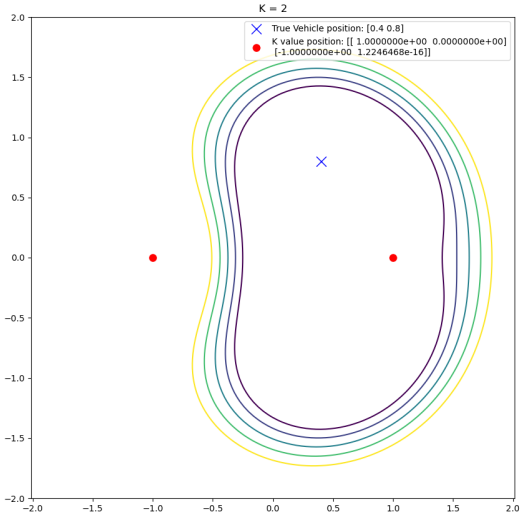
K = 2
True Vehicle position: [0.4 0.8]
K value position: [[1.0000000e+00 0.0000000e+00]
[-1.0000000e+00 1.2246468e-16]]
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Figure 12: K=2

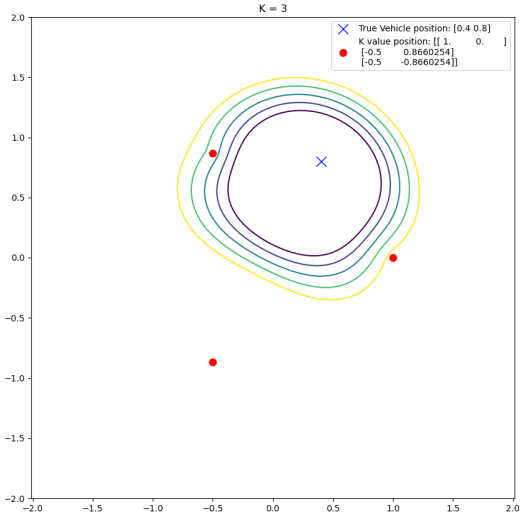
K = 3
True Vehicle position: [0.4 0.8]
K value position: [[1. 0.]
[-0.5 0.8660254]
[-0.5 -0.8660254]]
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Figure 13: K=3

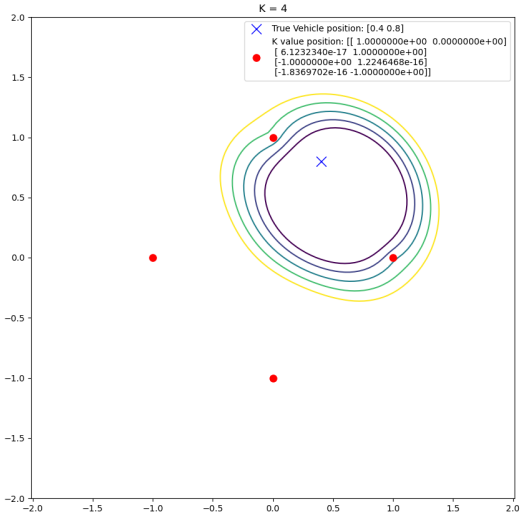

Figure 14: K=4

3 Question 4:

Problem 2.13 from Duda-Hart-Stork textbook:

Section 2.4

13. In many pattern classification problems one has the option either to assign the pattern to one of c classes, or to *reject* it as being unrecognizable. If the cost for rejects is not too high, rejection may be a desirable action. Let

$$\lambda(\alpha_i|\omega_j) = \begin{cases} 0 & i = j \qquad i, j = 1, \ldots, c \\ \lambda_r & i = c+1 \\ \lambda_s & \text{otherwise,} \end{cases}$$

where λ_r is the loss incurred for choosing the $(c+1)$th action, rejection, and λ_s is the loss incurred for making any substitution error. Show that the minimum risk is obtained if we decide ω_i if $P(\omega_i|\mathbf{x}) \geq P(\omega_j|\mathbf{x})$ for all j and if $P(\omega_i|\mathbf{x}) \geq 1 - \lambda_r/\lambda_s$, and reject otherwise. What happens if $\lambda_r = 0$? What happens if $\lambda_r > \lambda_s$?

The output of the question is from Page 22 and Page 26.

4 Question 5:

The ML estimator for θ is as follows:

$$p(z_n = k|\theta) = \theta_k, \qquad \prod_{k=1}^{K} \theta_k = 1$$

The MAP estimator, assuming that the prior $p(\theta)$ used for the parameters is a Dirichlet distribution with hyperparameter α, is as follows:

$$p(\theta|z,\alpha) \propto p(z|\theta)p(\theta|\alpha) \qquad = \prod_{n=1}^{N}\prod_{k=1}^{K} \theta_k^{z_{nk}} \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \quad \propto \prod_{k=1}^{K} \theta_k^{\sum_{n=1}^{N} z_{nk} + \alpha_k - 1}$$

This is the kernel of a Dirichlet distribution with parameter vector $\alpha' = (\alpha_1 + \sum_{n=1}^{N} z_{n1}, \ldots, \alpha_K + \sum_{n=1}^{N} z_{nK})$:

$$p(\theta|z,\alpha) = \frac{1}{B(\alpha')} \prod_{k=1}^{K} \theta_k^{\alpha_k + \sum_{n=1}^{N} z_{nk} - 1}$$

where $B(\alpha) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)}$ is the normalization constant of the Dirichlet distribution.

Question 1

$$\lambda_{ij} = \begin{cases} 0 & i = j \\ \lambda_r & i = c+1 \\ \lambda_s & \text{else} \end{cases}$$

$i, j = 1$ to c

Decisions: class	1	2	3	.....	n
1	0	λ_s	λ_s		λ_s
2	λ_s	0	λ_s		λ_s
3	λ_s	λ_s	0		λ_s
⋮					
n					0
n+1	λ_r	λ_r	λ_r		λ_r

In general, to maximize minimize risk →

$$\text{Compute } R(D = i/x) = \sum_{j=1}^{c} \lambda_{ij} P(L = j/x)$$

where

$R(D = i/x$ → cost associated in deciding label i

λ_{ij} → cost of (i, j)

$L = j/x$ → Posterior of label j.

$$\text{decision/action} = \underset{i}{\text{argmin}} \left(R(D = i/x) \right)$$

For our problem, considering classes i and k.

Risk associated in deciding class i →

$$R(D=i|x) = \sum_{j=1}^{c} \lambda_{ij} P(L=j|x)$$

$$= \lambda_s \left[\sum_{\substack{j \in \text{class labels} \\ j \neq i}} P(L=j|x) \right] \quad -(1)$$

where $\lambda_{ij} = 0$ when $j = i$ hence, that term has to be ignored.

$$\sum_{j=1}^{c} P(L=j|x) = 1$$

$$\therefore \quad \sum_{\substack{j \in \text{class labels} \\ j \neq i}} P(L=j|x) = 1 - P[L=i|x]$$

$\therefore$ equation (1) becomes.

$$R(D=i|x) = \lambda_s [1 - P[L=i|x]] \quad -(2)$$

Risk associated with deciding class k →

$$R(D=k|x) = \lambda_s [1 - P(L=k|x)] \quad -(3)$$

To decide between class i and class k →

Decide to class i if and only if →

$$R(D=i|x) < R(D=k|x)$$

Rephrasing →

$$R(D=i|K) \underset{D=i}{\overset{D=K}{\gtrless}} R(D=K|x)$$

Put in Substitute values of risk from equation ② & ③

$$\lambda_s [1 - P(L=i|x)] \underset{D=i}{\overset{D=K}{\gtrless}} \lambda_s [1 - P(L=K|x)]$$

$$P(L=i|x) \underset{D=K}{\overset{D=i}{\gtrless}} P(L=K|x)$$

Decide class i if $P(L=i|x) \geq P(L=K|x)$ for all class of K. $K = (1 \ldots c)$; $K \neq i$

* We still need to decide between class i and class $(c+1)$

Reject class → class $(c+1)$

Risk associated with reject class →

$$R[D=c+1|x] = \sum_{j=1}^{c} \lambda_{ij} P(L=j|x)$$

$$= \lambda_r \sum_{j=1}^{c} P(L=j|x)$$

$$= \lambda_r \rightarrow ④$$

Using eq^n (2) →

$$R[D=i|x] = \lambda_s [1 - P[L=i|x]]$$

Decision rule → minimize risk associated with decision.

∴ Decide class i if and only if →

$$R(D=i|x) \leq R(D=c+1|x)$$

$$\lambda_s [1 - P(L=i|x)] \leq \lambda_R$$

$$1 - P(L=i|x) \leq \lambda_R / \lambda_s$$

$$\Rightarrow \quad P(L=i|x) \geq (1 - \lambda_R/\lambda_s) \quad \rightarrow (5)$$

If $P(L=i|x) < (1 - \lambda_R/\lambda_s)$;

reject class $(c+1)$

(1) $\lambda_r = 0$ on eq^n (5)

$$P(L=i|x) \geq 1 - 0/\lambda_s$$

$$P(L=i|x) \geq 1$$

It is very rare to see that the decide class is i only if posterior ≥ 1.

Since $\lambda_r = 0$; cost associated with reject class is 0.

Hence, decision rule will always decide the reject class.

(2) $\lambda_r > \lambda_s$

Consider equation (5)

$$P(i=i|x) \geq 1 - \lambda_r / \lambda_s$$

If $\lambda_r > \lambda_s$: then

$$\boxed{1 - \frac{\lambda_r}{\lambda_s} < 0}$$

This is true for all probabilities of $[P(x) > 0]$

Decision rule will always choose some class i

It will never select reject class

This is intuitive as cost of reject class is high. if we wish to minimize risk.