Data Distribution

For C = 4 classes with uniform priors, parameters for the Gaussian Mixture Model:

$$\mu_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mu_1 = \begin{bmatrix} 2.5 \\ 0 \\ 0 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} 7.5 \\ 0 \\ 0 \end{bmatrix}$$

$$\Sigma_0 = \begin{bmatrix} 1 & 0.3 & 1.4 \\ 0.3 & 1 & 0.3 \\ 1.4 & 0.3 & 7 \end{bmatrix} \quad \Sigma_1 = \begin{bmatrix} 1 & -0.4 & -0.7 \\ -0.4 & 1 & -0.4 \\ -0.7 & -0.4 & 3 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 1 & 0.4 & 0.7 \\ 0.4 & 1 & 0.4 \\ 0.7 & 0.4 & 3 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 1 & -0.3 & -1.4 \\ -0.3 & 1 & -0.3 \\ -1.4 & -0.3 & 7 \end{bmatrix}$$

MLP Structure

`MLPClassifier` from `scikit-learn` `neural_network` module is utilized as it is modular, flexible, and easy to combine with other `scikit-learn` functions for creating and training neural networks for classification tasks.

- **2-layer MLP (one hidden layer of perceptrons):** `MLPClassifier` supports the creation of multilayer perceptrons (MLP) with customizable configurations for the number of hidden layers and the number of neurons in each layer. By default, the `MLPClassifier` is configured with a single hidden layer, but this configuration can be changed using the `hidden_layer_sizes` parameter. Setting the parameter `hidden_layer_sizes` to `[(p,) for p in counts]` enables the creation of an MLP with a single hidden layer containing `p` perceptrons, where `p` varies based on the values in the provided list `counts`. This allows for experimenting with different neural network architectures, as adjusting the `hidden_layer_sizes` parameter allows for the exploration of various model complexities (number of perceptrons) and the determination of the most effective structure through cross-validation.

- **Activation function:** The `activation` parameter of `MLPClassifier` enables the selection of an activation function for the hidden layers. `MLPClassifier` has limitations in the variety of activation functions it supports, notably not natively supporting smooth-ramp activation functions like ISRU, Smooth-ReLU, or ELU. The ReLU (Rectified Linear Unit) function is chosen for its computational efficiency and faster convergence in neural network training. `relu` is preferred because it involves simpler mathematical operations compared to `sigmoid` or `tanh` functions, resulting in faster computation during both the forward and backward passes of training. It also promotes sparse activation, where only a subset of perceptrons are activated in any given layer, enhancing computational and memory efficiency.

- **MLP output layer with softmax function:** In multi-class classification tasks, `MLPClassifier` automatically applies the softmax function at the output layer. This default behavior ensures that the output for each class is expressed as a probability and that the sum of these probability values for all classes equals 1 for any given input. This feature is important in multi-class classification as it supports the proper normalization of outputs, making them interpretable as probabilities.

Generate Data

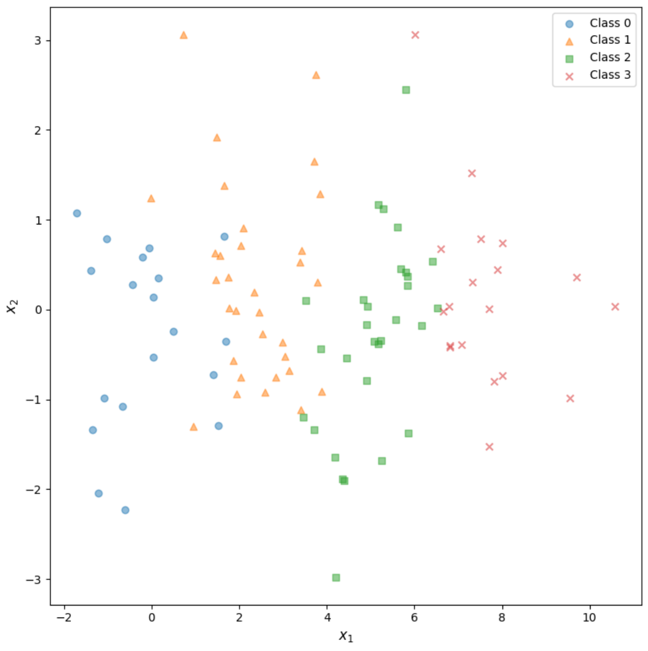


Figure 1: Training dataset with 100 samples

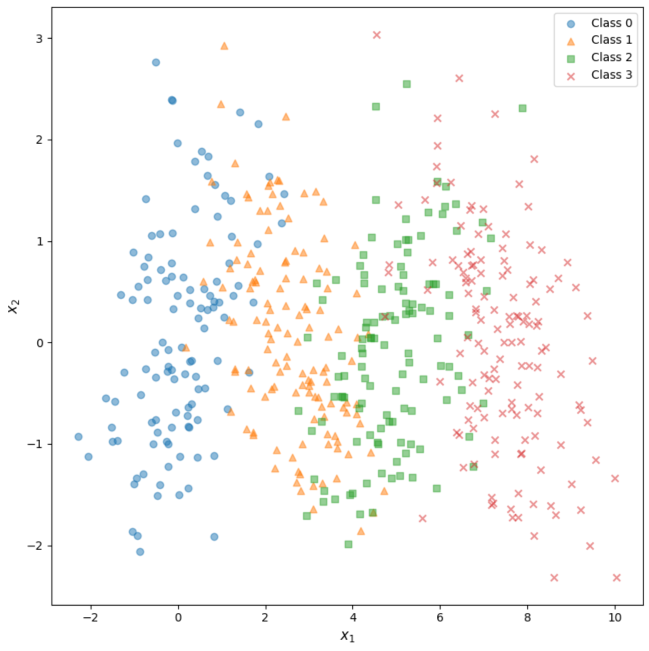


Figure 2: Training dataset with 500 samples

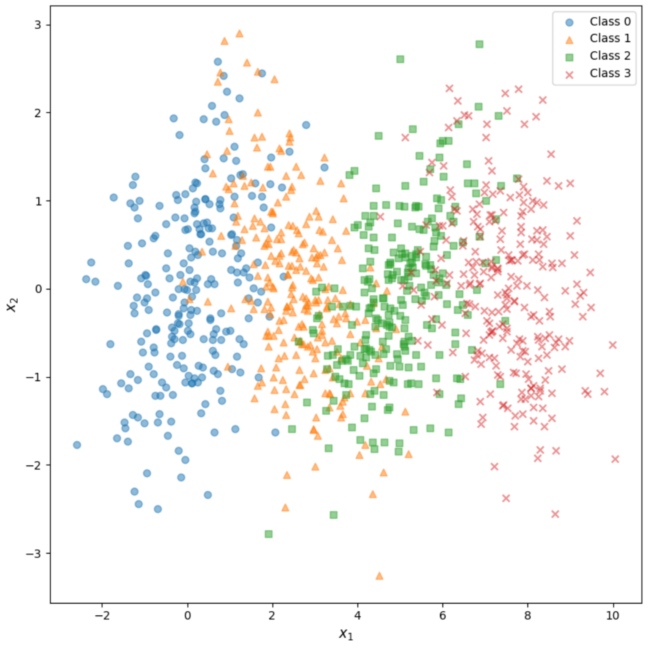


Figure 3: Training dataset with 1000 samples

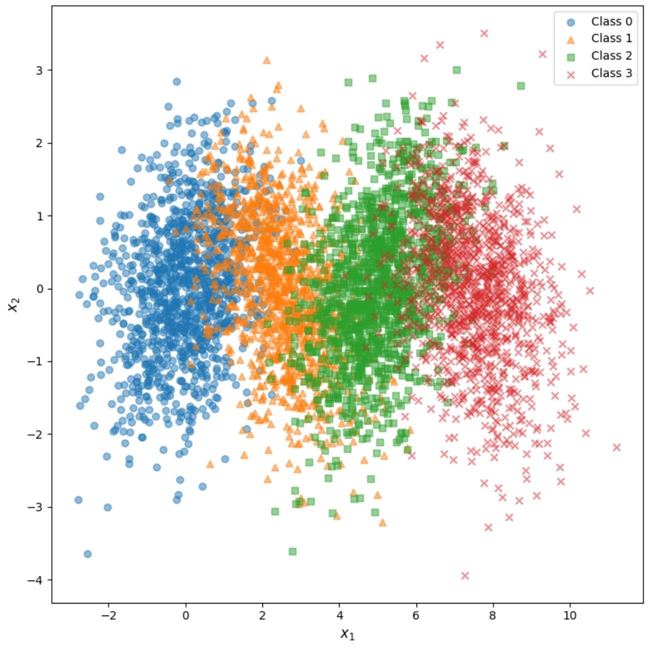


Figure 4: Training dataset with 5000 samples

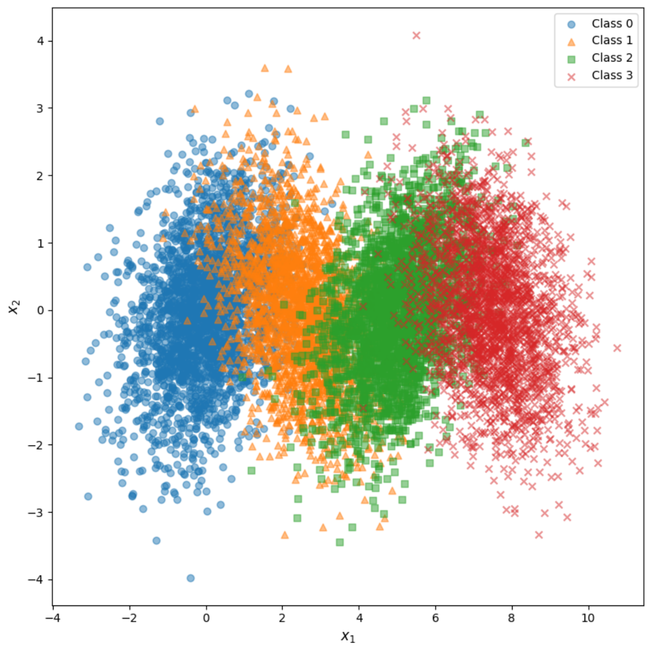


Figure 5: Training dataset with 10000 samples

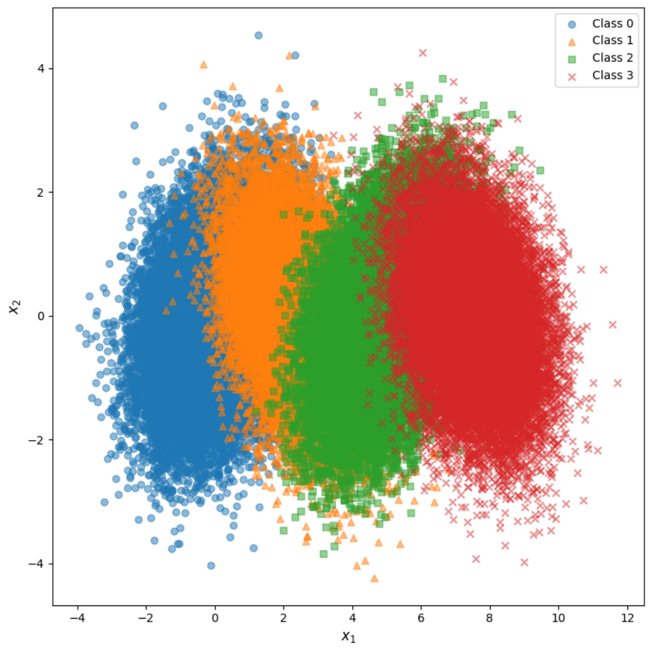


Figure 6: Validation dataset with 100000 samples

Theoretically Optimal Classifier

In the context of uniform prior probabilities, the minimum-probability-of-error classification MAP rule simplifies to the Maximum Likelihood (ML) rule. This approach involves calculating the likelihoods for each class using the Gaussian pdf, with class-specific mean vectors and covariance matrices. The decision rule then selects the class that maximizes the likelihood for each data point. The classifier's performance is assessed by comparing predicted and actual labels, determining the empirical probability of error. The probability of error on the test sample set using true data is approx. 13.35%. The Pr(error) value indicates the effectiveness of the selected parameters and validates the parameters as the classification rule for the true data pdf achieves between 10%–20% probability of error.

Model Order Selection

To determine the optimal number of perceptrons for each training set, K-fold cross-validation is utilized with the objective of minimizing classification error probability.This process involves the testing of various model configurations to identify the optimal one that minimizes the classification error probability. The process is implemented utilizing `GridSearchCV` from `scikit-learn`. `GridSearchCV` is a comprehensive tool for hyperparameter tuning which searches across a predefined grid of parameters, evaluating the performance of each model configuration through K-fold cross-validation.This provides an efficient framework for model selection and optimization, ensuring the most effective model structure is chosen.

- **Grid Search:** For models like `MLPClassifier`, `GridSearchCV` (Grid Search Cross-Validation) is utilized for hyperparameter tuning and model selection. It performs an exhaustive search over a predefined range of hyperparameters, testing multiple combinations to determine the optimal number of perceptrons. `GridSearchCV` assesses all possible combinations of perceptron counts across various model configurations. For each hyperparameter set, it fits a model to the training data and evaluates its performance using cross-validation. This approach ensures comprehensive exploration of the hyperparameter space and identification of the most effective model configuration based on the evaluation function.

  The grid search process optimizes the number of perceptrons and the regularization strength (alpha) based on the size of the training dataset. This approach is designed to align model complexity (number of perceptrons) with the volume of available data. Smaller datasets may lead to overfitting with overly complex models, while larger datasets might require more complex models to effectively capture patterns in the data.

  – **Perceptron Range:** The range of perceptrons tested for each MLP is chosen based on the size of the training set to balance model complexity with the number of data samples which is crucial for preventing overfitting in smaller datasets.

  - **Regularization:** The `alpha` parameter of `MLPClassifier` is a regularization term. A range of `alpha` values are evaluated during the grid search to determine the optimal level of regularization, aiming to minimize overfitting while allowing the model to learn the patterns in the data.

  These specific hyperparameter ranges ensure that the MLP is appropriately calibrated to the size of the training set, optimizing its ability to learn effectively without overfitting.The logarithmic spacing ensures a wide and balanced exploration of the regularization strength suitable for various data sizes.

  - $N \leq 500$:
    - No. of Perceptrons Range: [8, 16, 32] are selected to limit overfitting. Smaller models with fewer parameters are better suited for smaller datasets.
    - Alpha Range: np.logspace(-4, -2, 3) corresponds to $[10^{-4}, 10^{-3}, 10^{-2}]$. Higher regularization provides a stronger penalty against model complexity.
  - $500 < N \leq 2000$:
    - No. of Perceptrons Range: [32, 64, 128, 256] allow for greater model complexity.
    - Alpha Range: np.logspace(-5, -3, 3) corresponds to $[10^{-5}, 10^{-4}, 10^{-3}]$, providing slightly lower regularization strengths, enabling more complexity.
  - $N > 2000$:
    - No. of Perceptrons Range: [128, 256, 512] provide a wider range of complexity options.
    - Alpha Range: np.logspace(-6, -4, 3) corresponds to $[10^{-6}, 10^{-5}, 10^{-4}]$, further reducing the regularization strength for more flexibility.

- **Objective function:** The `scoring` parameter of `GridSearchCV` is utilized for determining how the performance of each model configuration is evaluated during cross-validation. To minimize the classification error probability, it's effective to inversely maximize accuracy, as error probability and accuracy are inversely related (Error Probability = 1 - Accuracy). By setting the `scoring` parameter to `accuracy`, `GridSearchCV` will evaluate and compare models based on their accuracy scores. This approach supports the objective of minimizing error probability via maximizing accuracy. Using `accuracy` as the scoring metric in `GridSearchCV` ensures that the model selection process is based on minimum classification error probability.

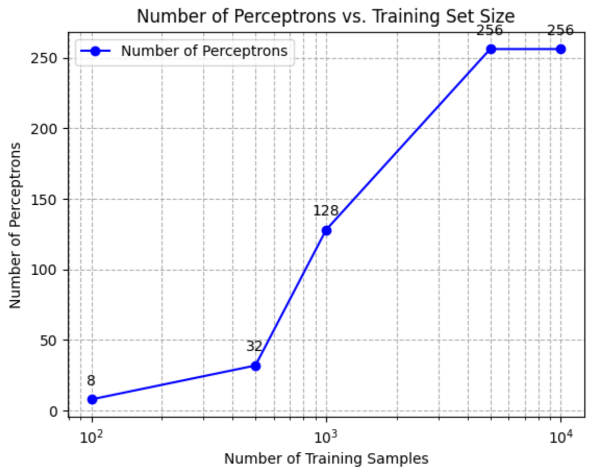
- **K-Fold Cross-Validation:** Evaluation of each hyperparameter combination in `GridSearchCV` is conducted through k-fold cross-validation instead of relying on a single train-test split. This method involves training and evaluating the model k times, each time using a different subset of the dataset for validation, ensuring the model's performance is validated across multiple subsets, improving the accuracy of the evaluation process and reducing the likelihood of overfitting. The performance evaluation metric for each model configuration is the average accuracy obtained across all k folds. The configuration that estimates the highest average accuracy indicating the lowest classification error probability is selected as the most optimal.

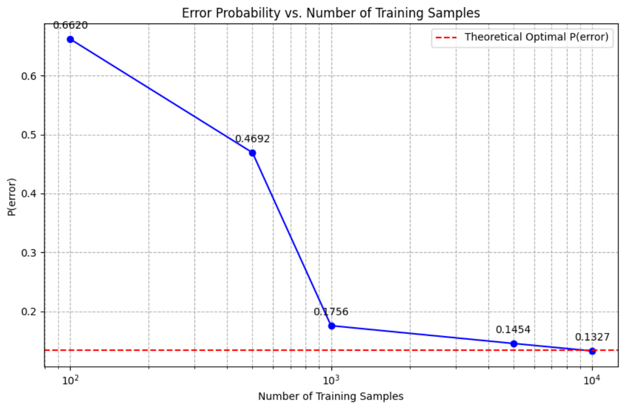
Analysis

Conducting three separate runs of model selection with the MLP on different datasets generated using the same parameters provides an optimized and comprehensive evaluation of the MLP's performance. This ensures that the chosen model is not overfitting to a specific dataset, but effectively capturing the general properties of the data distribution. Additionally, multiple executions also assesses the generalization ability of the MLP as the consistency of the MLP's performance across these varied runs highlights the reduced effect of the specifics of data generation such as random seed variations on performance.This confirms that the selected model is well-suited for diverse datasets derived from the same parameters.

P(Error) on test set using the true data pdf: 0.1337

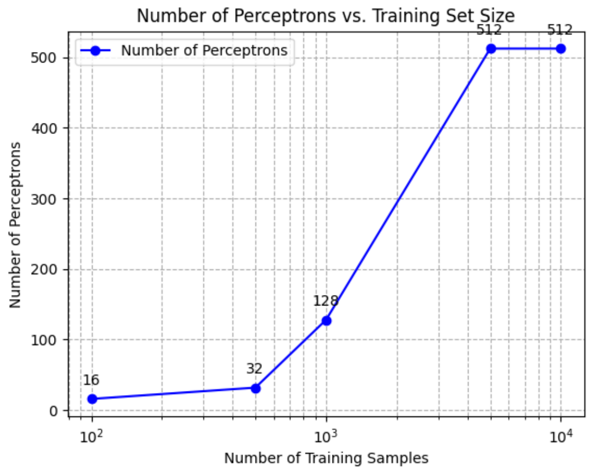
Training Set Size	**No. of Perceptrons**	**P(error)**
100	8	0.6620
500	32	0.4692
1000	128	0.1756
5000	256	0.1454
10000	256	0.1327

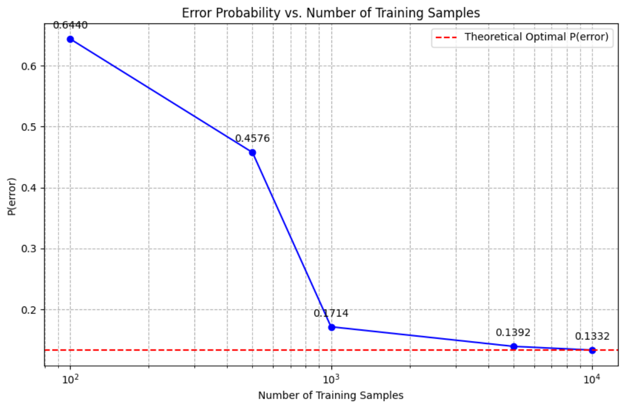




P(Error) on test set using the true data pdf: 0.1335

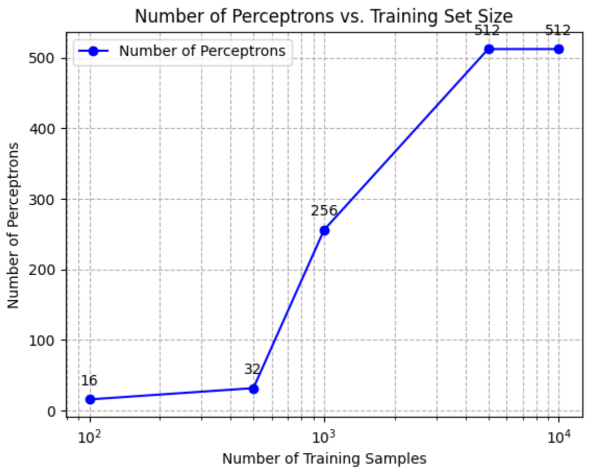
Training Set Size	No. of Perceptrons	P(error)
100	16	0.6440
500	32	0.4576
1000	128	0.1714
5000	512	0.1392
10000	512	0.1332

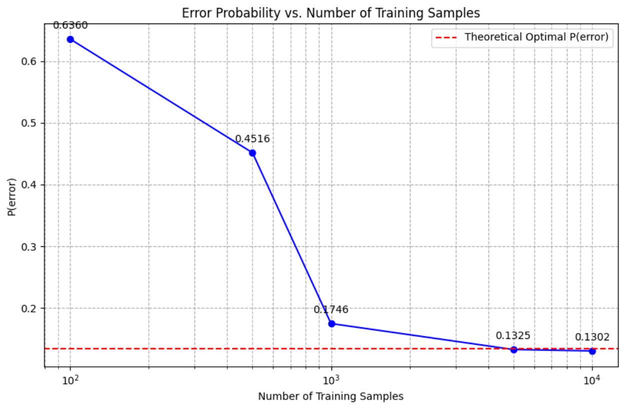




P(Error) on test set using the true data pdf: 0.1336

Training Set Size	No. of Perceptrons	P(error)
100	16	0.6360
500	32	0.4516
1000	256	0.1746
5000	512	0.1325
10000	512	0.1302





- **Accuracy improvement with more data:** As the dataset size increases, the error probability generally decreases. With a larger dataset, the model has greater capacity to accurately estimate the true data distribution, leading to more precise predictions. The overall decrease in error probability with an increase in training set size is an expected outcome as more data samples provide a more comprehensive representation of the data distribution, improving the learning algorithm's ability to generalize effectively.
- **Increasing model complexity for larger datasets:** For smaller datasets (100 and 500 samples), a lower number of perceptrons 8 and 32 is optimal. This indicates that a simpler model is sufficient for capturing the essential patterns in the data while avoiding overfitting. As the dataset size increases, there is a shift towards more complex models, with a higher number of perceptrons (128, 256, 512) being selected. This trend is expected and suggests the larger datasets provide enough data to support and validate the complexity of bigger models, effectively reducing the risk of overfitting. The trend of increasing model complexity (number of perceptrons) with increasing dataset size improves the model's learning capability while ensuring generalization.
- **Regularization and complexity control:** The decreasing trend in error probability with increasing training dataset size suggests that the regularization variation (via the `alpha` parameter of `MLPClassifier`) complexity variation for the number of perceptrons (via the `hidden_layer_sizes` parameter of `MLPClassifier`) according to number of data samples are working effectively. The models are complex enough to capture the necessary patterns but regularized to prevent overfitting.
- **Close proximity to theoretical optimal performance:** The error probability values, particularly for larger training sets, are closer to the theoretical p(error) of the optimally designed classifier, which is based on the true data pdf. This proximity in error rates indicates that the MPL configured with the selected hyperparameters, is efficiently approximating the optimal decision boundary as more data samples are available.The convergence of empirical probability of error towards the theoretical probability of error derived from the true data pdf highlights the effectiveness of the model tuning and the cross-validation process used in selecting the optimal level of model complexity.
- **Consistency across execution runs:** The consistency of error probability values across different runs with different seeds for the random number generation indicates stability in the model selection process. It suggests that the results are not based of particular random initializations or splits of the data.

Model Training

- **Maximum Likelihood Estimation:** MLE is used to estimate the model parameters (weights) that make the observed data most probable. In the context of neural networks, MLE aims to find the weights that essentially maximize the likelihood of observing the training data. In classification tasks, this translates into minimizing the cross-entropy loss, which is effectively the same as maximizing the log-likelihood of the correct labels.Cross-entropy loss measures how well the predicted probability distribution aligns with the true data distribution.A lower cross-entropy loss indicates better model performance as it suggests a smaller divergence between the predicted probabilities and the actual labels.

The MLP is trained using an optimization algorithm to minimize cross-entropy loss, effectively maximizing the log-likelihood of the training data under the model. Iterative reduction of cross-entropy loss using the optimization algorithm aligns the model parameters more closely with the underlying data distribution.

- **Optimization algorithm used for training:** The `solver` parameter of `MLPClassifier` specifies the optimization algorithm used for training the MLP. In neural network training, the solver essentially updates the weights of the network based on the loss function, which is the cross-entropy loss for classification tasks.

  The `solver` parameter is set to `adam`, indicating that the `adam` optimization algorithm is used for training. `adam` is an optimization algorithm that adjusts the learning rate for each weight in the neural network individually, resulting in more efficient weight updates, leading to faster convergence, especially in complex networks or large and variable datasets. It adjusts the weights iteratively based on the training data. By continuously updating the learning rates and utilizing the information from previous gradients, it minimizes the loss function more effectively compared to algorithms with a constant learning rate.

  This process of continuous adaptation and optimization ensures that the MLP effectively learns from the training data, with the goal of achieving the lowest possible cross-entropy loss (or highest log-likelihood).
- **Computing Log-Likelihood:** The `log_loss` function from `scikit-learn` computes and provides a measure of the cross-entropy loss, which is minimized during training. As the optimization algorithm used in training is designed to minimize rather than maximize cost function, maximizing log-likelihood is reformulated to minimizing the negative log-likelihood (cross-entropy loss). Minimizing negative log-likelihood is achieved by negating `log_loss`.A lower cross-entropy loss corresponds to a higher likelihood of the model accurately representing the observed data.Negating `log_loss` essentially allows likelihood maximization using optimization algorithm functions which minimize the cost function.

• **Mitigation of Local Optima in Training:** Neural network training aims to find the global minimum of a loss function which represents the optimal solution.In the context of MLP, there is complexity and potential for overfitting in multilayer perceptrons, which results in multiple local minima. These local minima can trap the training process, preventing it from reaching the global minimum.

  Random reinitialization is an effective strategy to overcome the limitations posed by local minima.The approach is to explore different parts of the parameter space (weights of the neural network) to improve the likelihood of finding a more globally optimal solution. The MLP training process is executed multiple times with different random initial values for the model's weights. Each initialization leads to a different trajectory through the parameter space during training and by exploring the parameter space, the training process is less likely to be stuck in local optima, increasing the chances of approaching the global minimum.

  After each training round, the performance of the resulting model is evaluated in terms of the log-likelihood of the training data.A higher log-likelihood implies a better fit to the training data, suggesting that the model has potentially reached a more optimal point in the parameter space.

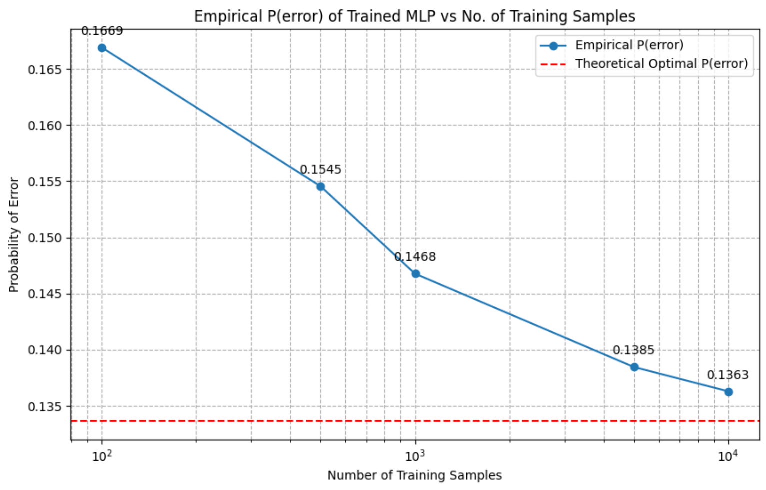
Performance Assessment

- **Estimation of class posterior probabilities:** `MLPClassifier` utilizes a softmax function to convert the output layer's scores into probabilities by normalizing these scores into a probabilistic distribution.This normalization ensures the output is interpretable as probability distributions over classes, allowing for a direct estimation of class posterior probabilities. The class posterior probabilities are the likelihoods of each class given the input features, conditioned on the current parameters (weights and biases at each layer adjusted during training) of the MLP.
- **Learning process:** During training, the MLP learns to approximate these posterior probabilities based on the distribution observed in the training data.The learning process is essentially an optimization problem, where the algorithm iteratively adjusts the parameters (weights and biases) through backpropagation and optimization algorithms (like sgd or adam) to minimize cross-entropy loss (difference between the predicted probabilities and the actual class distributions observed in the training data).
- **Generalization to unobserved data:** For validation test dataset, the trained MLP applies parameters learned during training to new, unseen data to estimate probabilities for each class. The ability of the model to accurately estimate class probabilities on test data indicates the accuracy of the MLP training and application of the learned information to classify the new data samples.

- **Application of MAP decision rule:** Post-training, the MLP functions as an estimator for the posterior probabilities of the classes. The `predict` function of `MLPClassifier` applies the MAP decision rule for classification, selecting the class with the highest posterior probability for each input sample.

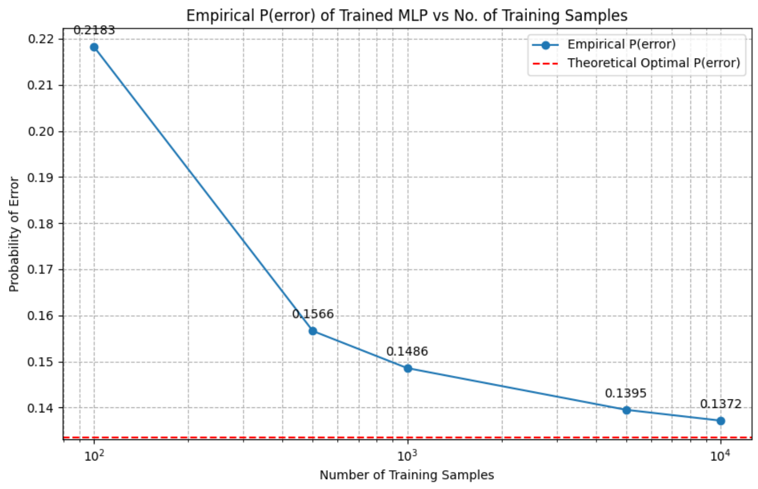
P(Error) on test dataset using the true data pdf: 0.1337

Training Set Size	P(error)	Accuracy
100	0.1669	0.8331
500	0.1545	0.8455
1000	0.1468	0.8532
5000	0.1385	0.8615
10000	0.1363	0.8637



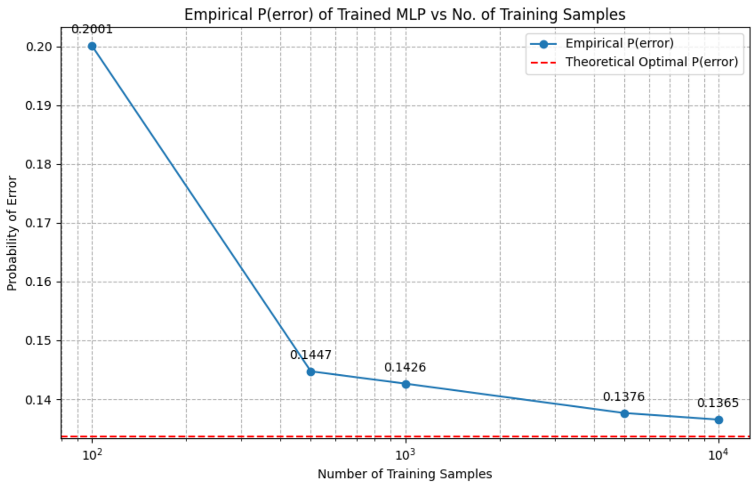
P(Error) on test dataset using the true data pdf: 0.1335

Training Set Size	P(error)	Accuracy
100	0.2183	0.7817
500	0.1566	0.8434
1000	0.1486	0.8514
5000	0.1395	0.8605
10000	0.1372	0.8628



P(Error) on test dataset using the true data pdf: 0.1336

Training Set Size	P(error)	Accuracy
100	0.2001	0.7999
500	0.1447	0.8553
1000	0.1426	0.8574
5000	0.1376	0.8624
10000	0.1365	0.8635



- **Decreasing probability of error with increasing training dataset size:** Expected observation that the error probability in machine learning models, including MLPs, will generally decrease as the number of training data samples increases. This aligns with the theoretical understanding that more data samples provides a better representation of the data distribution, allowing the model to develop a more accurate and reliable mapping from inputs to outputs. Additionally, the increased diversity in the data samples may assist in reducing biases and overfitting, leading to a more optimal and accurate model.

- **Performance variability with fewer samples:** Models trained with only 100 data samples show a higher variability in performance, with probability of error values ranging from 0.1669 to 0.2183. This variability is due to the higher influence of random variation and noise in limited training data, potentially leading to overfitting or underfitting.

- **Training dataset size vs. model complexity:** The complexity (number of perceptrons) of the MLP is optimally chosen relative to the size of the training set to avoid overfitting.This optimal selection is highlighted as the models do not exhibit a significant increase in error probability when transitioning from the training dataset to the test dataset. This suggests that the MLP's were not overfit to the training data during training. This suggests that the number of perceptrons is optimal allowing the model to effectively learn the patterns during training.

- **Model generalization:** The models demonstrate good generalization capabilities, especially those trained with larger datasets. This is highlighted by the error probability values closely aligning with theoretical error probability indicating that the models are not overfitted to the training data and are capable of generalizing well to the new, unseen test dataset.