EECE5644 2022 Summer 1 – Assignment 3

Submit: Thursday, 2022-June-16 before 11:59 EDT (upload PDF to Canvas)

Please submit your solutions at the classroom assignments page in Canvas in the form of a single PDF file that includes all math, numerical and visual results. Also, for verification of the existence of your own computer implementation, include a link to your online code repository or include the code as an appendix / attachment in a ZIP file along with the PDF. The code is not graded, but helps verify your results are feasible as claimed. Only results and discussion presented in the PDF will be graded, so do not link to an external location where further results may be presented.

This is a graded assignment and the entirety of your submission must contain only your own work. You may benefit from publicly available literature including software (not from classmates), as long as these sources are properly acknowledged in your submission. All discussions and materials shared during office periods are also acceptable resources and these tend to be very useful, so participate in office periods or take a look at their recordings. Cite your sources as appropriate. Discussing verbally with classmates are acceptable, but there can not be any written material exchange.

By submitting a PDF file in response to this take home assignment you are declaring that the contents of your submission, and the associated code is your own work, except as noted in your citations to resources and allowed otherwise as described.

Question 1 (60%)

In this exercise, you will train many multilayer perceptrons (MLP) to approximate the class label posteriors, using maximum likelihood parameter estimation (equivalently, with minimum average cross-entropy loss) to train the MLP. Then, you will use the trained models to approximate a MAP classification rule in an attempt to achieve minimum probability of error (i.e. to minimize expected loss with 0-1 loss assignments to correct-incorrect decisions).

Data Distribution: For $C = 4$ classes with uniform priors, specify Gaussian class-conditional pdfs for a 3-dimensional real-valued random vector $\mathbf{x}$ (pick your own mean vectors and covariance matrices for each class). Try to adjust the parameters of the data distribution so that the MAP classifier that uses the true data pdf achieves between $10\% - 20\%$ probability of error.

MLP Structure: Use a 2-layer MLP (one hidden layer of perceptrons) that has P perceptrons in the first (hidden) layer with smooth-ramp style activation functions (e.g., Smooth-ReLU, ELU, etc)[1]. At the second/output layer use a softmax function to ensure all outputs are positive and add up to 1. The best number of perceptrons for your custom problem will be selected using cross-validation.

Generate Data: Using your specified data distribution, generate multiple datasets: Training datasets with $100, 200, 500, 1000, 2000, 5000$ samples and a test dataset with 100000 samples. You will use the test dataset only for performance evaluation.

Theoretically Optimal Classifier: Using the knowledge of your true data pdf, construct the minimum-probability-of-error classification rule, apply it on the test dataset, and empirically estimate the probability of error for this theoretically optimal classifier. This provides the aspirational performance level for the MLP classifier.

Model Order Selection: For each of the training sets with different number of samples, perform 10-fold cross-validation, using minimum classification error probability as the objective function, to select the best number of perceptrons (that is justified by available training data).

Model Training: For each training set, having identified the best number of perceptrons using cross-validation, using maximum likelihood parameter estimation (minimum cross-entropy loss) train an MLP using each training set with as many perceptrons as you have identified as optimal for that training set. These are your final trained MLP models for class posteriors (possibly each with different number of perceptrons and different weights). Make sure to mitigate the chances of getting stuck at a local optimum by randomly reinitializing each MLP training routine multiple times and getting the highest training-data log-likelihood solution you encounter.

Performance Assessment: Using each trained MLP as a model for class posteriors, and using the MAP decision rule (aiming to minimize the probability of error) classify the samples in the test set and for each trained MLP empirically estimate the probability of error.

Report Process and Results: Describe your process of developing the solution; numerically and visually report the test set empirical probability of error estimates for the theoretically optimal and multiple trained MLP classifiers. For instance show a plot of the empirically estimated test P(error) for each trained MLP versus number of training samples used in optimizing it (with a semi-log plot, logarithmic scale on the x-axis), as well as a horizontal line that runs across the plot indicating the empirically estimated test P(error) for the theoretically optimal classifier.

[1] Feel free to use ReLU as well, Smooth-ReLU or "softplus" is less efficient and effective in practice. See `https://deeplearninguniversity.com/pytorch/pytorch-activation-functions/` for a list of activation functions implemented in PyTorch.

Note: You may use software packages for all aspects of your implementation, I especially recommend PyTorch for its ease of use. Make sure you use tools correctly. Explain in your report how you ensured the software tools do exactly what you need them to do.

Question 2 (40%)

We will derive the solution for ridge regression in the case of a linear model with additive white Gaussian noise corrupting both input and output data. Then we will implement it and select the hyper parameter (prior parameter) with cross-validation. Use $n = 10$, $N_{train} = 50$, and $N_{test} = 1000$ in this question.

Generate Data: Select an arbitrary non-zero n-dimensional vector $\boldsymbol{a}$. Pick an arbitrary Gaussian with nonzero-mean $\boldsymbol{\mu}$ and non-diagonal covariance matrix $\boldsymbol{\Sigma}$ for a n-dimensional random vector $\boldsymbol{x}$. Draw N_{train} iid samples of n-dimensional samples of $\boldsymbol{x}$ from this Gaussian pdf. Draw N_{train} iid samples of a n-dimensional random variable $\mathbf{z}$ from a $\mathbf{0}$-mean $\alpha\boldsymbol{I}$-covariance-matrix Gaussian pdf. Draw N_{train} iid samples of a scalar random variable v from a 0-mean unit-variance Gaussian pdf. Calculate N_{train} scalar values of a new random variable as follows $y = \boldsymbol{a}^T(\boldsymbol{x}+\boldsymbol{z})+v$ using the samples of $\boldsymbol{x}$ and v. This is your training dataset that consists of $(\boldsymbol{x}, y)$ pairs. Similarly, generate a separate test dataset that consists of N_{test} $(\boldsymbol{x}, y)$ pairs.

Model Parameter Estimation: We think that the relationship between $(\boldsymbol{x}, y)$ pairs is linear $y = \boldsymbol{w}^T\boldsymbol{x} + w_0 + v$ and v is an additive white Gaussian noise (with zero-mean and unit-variance). We are unaware of the presence of the noise term $\boldsymbol{z}$ in the true generative process. We think that this process also has linear model parameters close to zero, so we use a 0-mean and $\beta\boldsymbol{I}$-covariance-matrix Gaussian pdf as a prior for the model parameters $\boldsymbol{w}$ (which contain w_0). We will use MAP parameter estimation to determine the optimal weights for this model using the training data.

Hyper-parameter Optimization: The prior for the $n+1$-dimensional weight vector in the model has a scalar parameter β that needs to be selected. Use 5-fold cross-validation on the training set to select this parameter. As your cross-validation objective function, use max-log-likelihood of the validation data, averaged over the 10 partitions. Use the MAP parameter estimation solution in the process with candidate hyper-parameter values.

Model Optimization: Once the best hyper-parameter value is identified, use the entire training dataset to optimize the model parameters with MAP parameter estimation that specifically uses this best hyper-parameter selection. Evaluate the '-2 times log likelihood' of the test data with the trained model.

While keeping $\boldsymbol{a}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ constant, vary the noise parameter α gradually from very small (e.g., $10^{-3}trace(\boldsymbol{\Sigma})/n$) to very large (e.g., $10^{3}trace(\boldsymbol{\Sigma})/n$), introducing increasing levels of noise to the input variable, generate new training and test datasets with each α, repeat the process, and analyze the impact of this noise parameter on the solution, selection of the hyperparameter, and performance on the test set.

Show your math for the derivation of the MAP parameter estimate, explain how your code implements the solution, and present numerical and visual results, along with their discussions.