Assignment 3

Neel Bhalla

November 2021

1 Intro

The Majority of code was writetn in Python 3.7 (with a single file in matlab r2020a). Data is stored in folders `q1data/` and `q2data/`.
Code can be found at
`https://github.com/ashdawngary/EE5644/tree/main/A3`

2 Question 1

2.1 Selecting The Gaussian Mixture

As per request of the problem, A uniform prior, 4-class Gaussian mixture was created such that the minimum p-error of the classifier given the true PDF was between 10% and 20%.

The tetrahedron case was chosen (utilized in A1), with vertices at (1,1,1), (1,-1,-1), (-1, 1, -1), and (-1,-1,1). The covariances for each of the classes were γI for some γ. Next the choice of γ (variance) was plotted against min-perror in order to find a suitable value that lied within our tolerances:

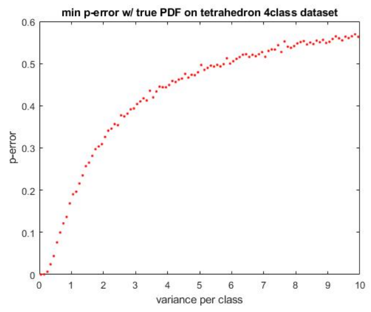


Figure 1: probability of errors with classifier based on True PDF parametrized by data set class variance γ

It follows from the tolerances that $\gamma = 1$ is a suitable selection that yields a approximate p error of .178 for a optimal classifier (given true PDF).

Next data was generated using I for covariance, means listed above, and priors of $\frac{1}{4}$ each. Training sets of sizes 100, 200, 500, 1000, 2000, and 5000 were created with a test set of size 100000. These datasets are encoded in numpy binary files within `q1data/`.

2.2 Structure of network

The structure of the network is generally a dense layer of size p (dense meaning all pairwise connections possible between layers), which is linked to another dense layer of 4. After the p layer, there is a nonlinear ELU activation function, and the output of the 4 width Dense layer is a softmax, in order to convert the outputs into something interpretable as a probability distribution.

2.2.1 ELU Activation

ELU(Exponential Linear Unit), unlike ReLU, is a piece-wise smooth nonlinear function that is used as an activation function in this context. Unlike RElu which can be expressed as $\max(0, x)$, ELU is defined as

$$\begin{cases} \alpha(e^x - 1) & x \leq 0 \\ x & x > 0 \end{cases}$$

In order to make ELU piecewise smooth, the default value of $\alpha = 1$ makes sure that both sides of the function at $x = 0$ have a derivative of 1.

2.2.2 Softmax Activation

The softmax activation function is an activation function that transforms a vector (unlike ELU) into a new vector such that $||x||_1 = 1$. More specifically Softmax(x) is defined as $\frac{1}{\sum_{i=1}^{N} e^{x_i}} * exp(x)$, where exp is a element-wise operation $x \to e^x$

The choice of what is the best value of k for the network is decided through the experiment detailed in this section which utilizes k-fold validation.

2.3 Training Techniques

For a given p, training the network consists of iterative numerical optimization to minimize difference between the one hot vector of a label and the output of the network. (OneHot(x) = A vector with zeros, but a 1 at position x).

In order to prevent overfitting of the model to the data, the k-fold procedure is implemented on top of the training procedure.

2.4 K fold validation

K-fold validation consists of creating a 'robuster' metric of model success by not testing the final output on the data it was trained on. Given the original training data D_t, and a fold factor k, the data is split into k almost equal partitions. Then k experiments are produced where experiment i consists of treating parition i as *validation* data, whereas the rest is *training* data.

Numerical Optimimzation methods (eg: stochastic gradient descent) is performed on the training data. The performance on the model for the sakes of k-fold is measured using the validation data. Since the model has not encountered these samples before, it is not possible to overfit the model to the truth label of these values. Finally taking the average heuristic (or weighted by proporition of validation data), provides a robust outlook on the performance of the general model.

In this case, k was chosen as 10, and was performed to determine how effective using p perceptrons was for the hidden layer. Then kfold experiments were performed starting with $p = 1$ and increasing until the change in probability of error between p values was less than 5% over 5 values of p. This indicates to us that we have reached a stale point, and increasing p does not have a significant effect.

Other heuristics include performing statistical significance tests between continuous ranges to determine difference in performance. Finally an arbitrary max perceptron limit of 20 was introduced to put a hard limit on how many trials could be run.

2.5 Loss Function

Since we are working with classification data, the Catagorical Cross-Entropy loss function was chosen. This function is an extension of the binary cross-entropy loss function described in class for multiple class labels.

2.6 Results

2.6.1 Packages

All the Code was implemented with keras which uses tensor-flow for numerical calculations / gradient calculations.

Within Keras, it suffices to create a model which is passed in a list of layers, dense p with activation function elu, default $\alpha = 1$, and then a dense 4 layer with actvation softmax. Simmiarily, given a softmaxed output layer, the loss function from Keras `keras.losses.CategoricalCrossentropy()` will calculate the loss for the model. Finally the optimizer is the adam optimizer, which dynamically changes the step size for stochastic gradient descent to converge without ”jumping around” the minima for too long.

2.6.2 minimum p error over data samples

One result derived from the experiment performed is that the minimum probability of error decreases as we add more points to our dataset. While we are bounded below by a theoretical optimal (given the true PDF), adding more points allows for the network to better approximate the true pdf, thus we achieve closer to theoretical optimal.

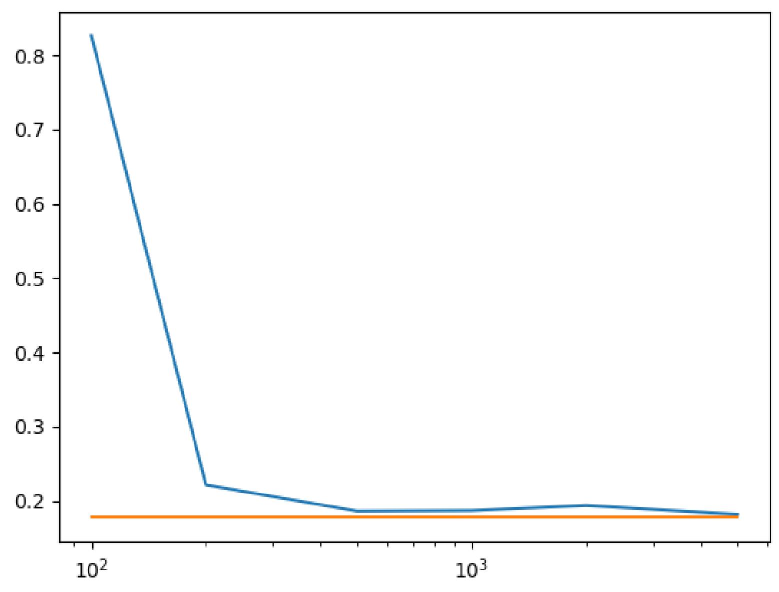


Figure 2: Graph of probability of error (kfold) vs number of training samples. Orange line is the theoretical optimal classifier probability of error (approx .178)

It follows form the graph that 10-fold validation with 100 samples is not sufficient, but adding more samples allows the network better approximate the true labels. By 1000 training samples, the classifier has enough to reach the theoretical optimal bound.

2.6.3 Rate Convergence wrt with different datasets

Another trend that was observed was that the choice of p (number of perceptrons) inversely varies with the size of the dataset. Granted for any given p, adding more datapoints will yield a classifier which does at least as better as originally (It shouldnt do worse!), but it turns out that smaller values of p become more viable. These findings are summarized in the figure below:

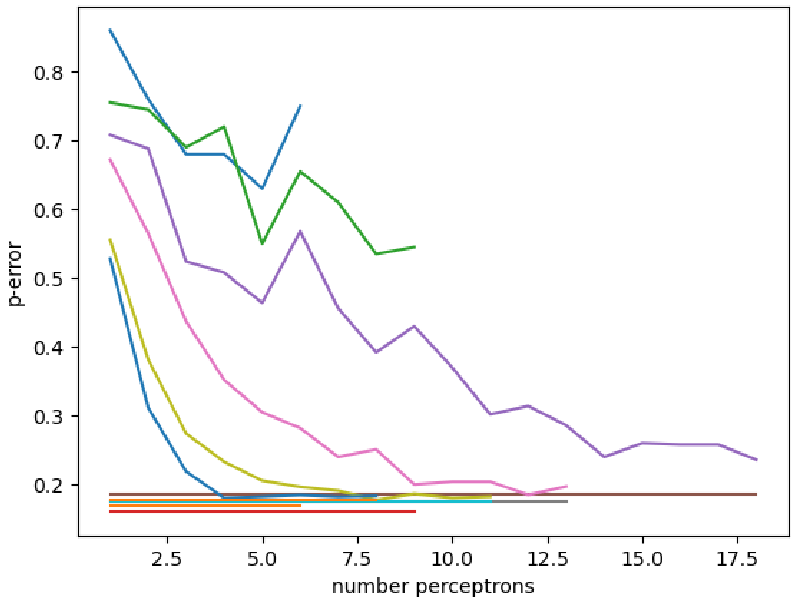


Figure 3: P error wrt number of perceptrons. From top to bottom: Blue: 100 samples, Green: 200 samples, Purple: 500 samples, Pink: 1000 samples, Yellow: 2000 samples, Blue: 5000 samples. 100 and 200 samples terminated early since they met convergence criteria(5% rel decrease per 5 epochs)

For 100 datasamples, a p of 6 was chosen, but it clearly does not perform well. Should we have relaxed the criteria further, it may decrease with number of perceptrons, but it is believed that there is not enough training data to help it reach the theoretical lower bound. As for 500 samples, it reached the theroetical lower bound with 18 perceptrons, but as we increase the number of datapoints to 5000, the theoretical optimal classifier is reached by 8 perceptrons!

3 Question 2

In this question, k fold expierments were utilized to determine the best model order for a distribution of data. In specific, the model order refers to the number of gaussians in the mixture of the output PDF. Since data fitting is a data sensitive task, 30 sets of training data were generated(for each 10, 100, 1000, and 10000 samples), and model orders were predicted for each experiment.

3.1 Software packages

In order to access the EM algorithm for fitting gaussians (given a predicted number of components), `scikit-learn`'s `GaussianMixture` class was employed. It contains a `fit` function to run EM to convergence, and contains a `score_samples` function to compute log likelyhood for each sample given. These were sued to fit the gaussian and score it via validation data.

3.2 Distribution selection

The gaussian mixture is composed of 4 components. The priors for the components are 0.15, 0.35, 0.2 and 0.3. The mean vectors are $\langle -2, -1\rangle$, $\langle 2, 2\rangle$, $\langle 0, -2\rangle$, and $\langle -2, 2\rangle$. The covariances are $c(0.25, 0.5)$, $c(1, 1)$, $c(0.5, 1)$, and $c(0.75, 0.25)$ where $c(x, y)$ is $\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}$. Below is a plot of the data:

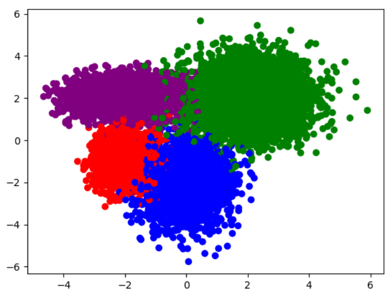


Figure 4: 10000 samples of the distribution

Data was sampled 30 times for the 30 experiments. Data is stored in raw numpy binary files in the `q2data/` folder.

3.3 K fold objective

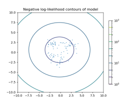
Unlike before where it sufficed to take the heuristic has the probability of error, in this case we need to measure how well does the model's pdf match the validation data. This is done by summing the log likelihood of the validation set on the model. Since the model has not seen these data-points before, the model that best approximates the distribution will maximize the probability of these points.

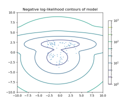
We take the negative of this quantity to make it a minimization problem for the model order.

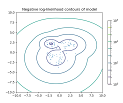
3.4 Results

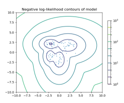
3.4.1 Fitting Results

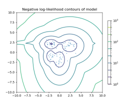
One observation made was that as datapoints are added, the quality of fit increases for every model order. Below are the pdf's of the Gaussian mixtures for 100 samples: When compared to the results for 10000 samples:


(a) 1


(b) 2


(c) 3


(d) 4


(e) 5

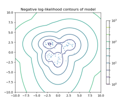
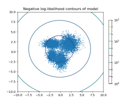
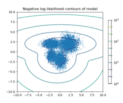
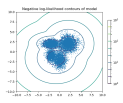
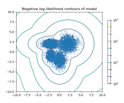

(f) 6

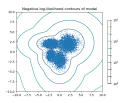
Figure 5: Gaussian Mixture pdf's for experiment 1 with 100 samples. It is clear that by 3 components it seems to have captured the essence of the data. Extra Gaussian's seem to be over-fitting to the data, as reflected in the selection matrix below. Viewable in `A3/q2/modelfits/10/*`


(a) 1


(b) 2


(c) 3


(d) 4


(e) 5

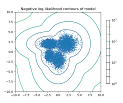

(f) 6

Figure 6: Gaussian Mixture pdf's for experiment 1 with 10000 samples. It is clear that 4 components are required instead of 3 prior.Viewable in `A3/q2/modelfits/10000/*`

3.5 Model Selection Results

The Matrix below captures the essence of 30 experiments run.

```
[[29.   1.   0.   0.   0.   0.]
 [ 0.   0. 10. 18.   2.   0.]
 [ 0.   0.   0. 17. 12.   1.]
 [ 0.   0.   0. 21.   4.   5.]]
```

The rows correspond to sizes of dataset (10, 100, 1000, and 10000), whereas columns represent model orders (1,2,3,4,5,6). Every row must add up to 30(number of trials). In an ideal setting, the matrix would be all zeros, except for the 4th column which would be all 30s, but due to data sparsity, it is not clear which model order is appropriate for smaller models.

For example, for 10 samples, 10 fold cross validation does not make any sense. There is exactly **1** point in the validation dataset, which is not enough to determine the simmilarity of the distribution to the true pdf. As a result of the flawed heuristic, fitting the model better to the 9 other points yielded a lower log likelyhood for the validation point, thus model order 1 was consistently selected.

3.5.1 Probability of Correct Selection

As we increase the number of samples, the probability of selecting the true model order increases. It is interesting that in this run of 30 experiments, the 100 dataset correctly predicted the model order better than the 1000 dataset, but in *general*, this is not the case(it was an anomaly, but it takes approx 10 minutes to run these experiments, hence expensive to regenerate).

3.5.2 Tendency of Error

A consistent observation is the tendency of errors. In the 10 and 100 datasets, the model order tends to be pessemistically chosen, favoring errors of smaller model order over over compensating. For example, model order 4 was decided the majority of the time in 100 sample dataset, but it reported a lower model order 10/30 times whereas it reported a higher model order only 2 times. On the other hand as we increase the dataset sizes to 1000/10000, it never reported a lower model order than 4, but did report model orders higher.

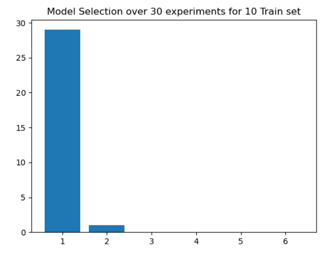


Figure 7: Distribution of model order selection over 30 exp for 10 samples.

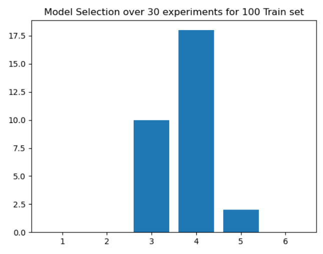


Figure 8: Distribution of model order selection over 30 exp for 100 samples.

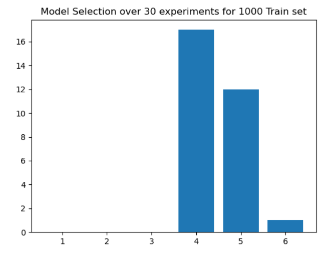


Figure 9: Distribution of model order selection over 30 exp for 1000 samples.

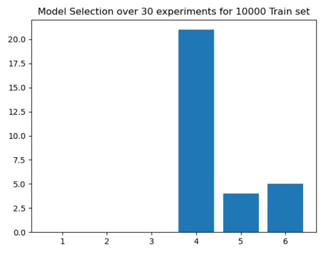


Figure 10: Distribution of model order selection over 30 exp for 10000 samples.