EECE 5644 Assignment 2

Eric Grimaldi

2020 November 5

Foreward on Code Libraries

For reference:

The code for this assignment was completed in *Python*. The *NumPy* library is used extensively for linear algebra and random variables. The *SciPy* library is used for a few linear algebra functions with improved numerical calculation. *SciPy* also provides the *scipy.optimize.minimize*() function used for Question 1 Part 3. The *MatPlotLib* is used for data visualization. *SciKitLearn* provides the $sklearn.mixture.GaussianMixture().fit()$ function used for quickly completing Expectation Maximization estimations after an initial implementation of the algorithm.

Question 1

As per the assignment:

X is a two-dimensional random vector with the PDF below:

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

$L = l$ signifies the true class label. The class priors are $P(L = 0) = 0.6$ and $P(L = 1) = 0.4$. The class-conditional PDFs are $p(x|L = 0) = w_1 g(x|m_0 1, C_0 1) + w_2 g(x|m02, C02)$ and $p(x|L = 1) = g(x|m_1, C_1)$ where $g(x|m, C)$ is a multivariate Gaussian PDF with mean m and covariance matrix C. The parameters in this case are

$$m_{01} = \begin{bmatrix}5\\0\end{bmatrix} C_{01} = \begin{bmatrix}4 & 0\\0 & 2\end{bmatrix} m_{02} = \begin{bmatrix}0\\4\end{bmatrix} C_{02} = \begin{bmatrix}1 & 0\\0 & 3\end{bmatrix} m_1 = \begin{bmatrix}3\\2\end{bmatrix} C_1 = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}$$

For numerical results below, we generate 4 sets of data according to this distribution:

- D_{train}^{100} consisting of 100 samples and their labels, used for training
- D_{train}^{1000} consisting of 1000 samples and their labels, used for training
- D_{train}^{10000} consisting of 10000 samples and their labels, used for training
- $D_{validate}^{20000}$ consisting of 20000 samples and their labels, used for validation

To draw these samples, we first randomly choose a true label according to the class priors - if $L = 0$ is chosen, then we further randomly choose between g_{01} and g_{02} according to weights w_1 and w_2 - and then randomly draw a sample data point from the appropriate class-conditional PDF by using *numpy.random.multivariate_normal*(). All parts of the question are completed with a single data set in a single run of the code.

Part 1: Classification with Knowledge of True PDF

As provided in $L02a\_ExpectedRiskMinimizationBasedClassifierDesign.pdf$:

The likelihood-ratio test (LRT) for the 2-class case of Expected Risk Minimization (ERM) is as follows:

$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{\lambda_{10}-\lambda_{00}}{\lambda_{01}-\lambda_{11}} \frac{P(L=0)}{P(L=1)} = \gamma$$

where λ_{ij} is the loss value for $(D = i|L = j)$. For the minimum expected risk case, λ_{ij} is 0 $\forall i = j$ and is 1 $\forall i \neq j$. This reduces the LRT to :

$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{P(L=0)}{P(L=1)} = \gamma$$

After full substitution and simplification, this becomes:

$$\frac{|C_1|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-m_1)^T C_1^{-1}(x-m_1)}}{\frac{1}{2}|C_{01}|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-m_{01})^T C_{01}^{-1}(x-m_{01})} + \frac{1}{2}|C_{02}|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-m_{02})^T C_{02}^{-1}(x-m_{02})}} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{0.6}{0.4} = 1.5 = \gamma$$

As per the assignment, we determine the theoretically optimal γ to be 1.5. We generate an ROC curve for this classifier by plotting True Positive Rate vs False Positive Rate as we vary γ slowly from 0 to ∞. Since we have a finite number of data points in $D_{validate}^{20000}$, we are able to choose a finite number of γ_i in order to accomplish this. We choose 20001 γ such that $\gamma_0 = 0$, γ_i is halfway between the result of the ith LRT and the $(i+1)$th LRT, and the final γ_{10000} is double the 10000th LRT for good measure.

For each γ_i we classify the entire data set, keeping track of the decision-label pairs, and calculating the confusion matrix quantities for each γ_i (True Positive rate, False Positive rate, True Negative rate, False Negative rate). Using the False Positive and True Positive rates across all γ_i as (x, y) data we plot the ROC curve for this classifier. See $Figure1$.

We also empirically determine $\gamma_{\hat{P}_e}$ which minimizes the probability of error by simply keeping track of the lowest P_e as we move through γ for the ROC curve. $\gamma_{\hat{P}_e}$ is superimposed on the ROC curve, as you can see in $Figure1$, along with the relevant $\hat{P}_e$, TP, and TN.

The exact $\gamma_{\hat{P}_e}$ and $\hat{P}_e$ varies slightly depending on the data set generated. Typically $\gamma_{\hat{P}_e}$ achieves a value very close to the theoretical optimum of 1.5. In the run recorded for this report, we see $\gamma_{\hat{P}_e}$ of 1.553 and $\hat{P}_e$ of 17.20%. Compared to our previous assignment, this error rate seems high, however in this case one of the labels is split across two Gaussians on either side of a Gaussian for the other label; one can imagine points near the middle where the combined likelihood of the outer two Gaussians and the likelihood of the inner Gaussian become similar, and thus difficult to distinguish. Further, some of the covariance values lead to a bit more overlap than the previous assignment.

Part 2: Classification with PDF Parameters Estimated via ML

We repeat the previous analysis with PDF parameters estimated via the maximum likelihood parameter estimation technique.

The class priors $P(L=0)$ and $P(L=1)$ are both estimated by a simple ratio:

$$P(L=l) = \frac{N_l}{N}$$

Where $L=l$ is the true class label, N_l is the number of $L=l$ occurrences in D_{train}, and N is the total number of samples in D_{train}.

Since $L=1$ is represented by a single Gaussian, the ML estimation for m_1 and C_1 both reduce to a sample averages:

$$\hat{m}_{1ML} = argmin_{m_1} \frac{1}{N} \Sigma_{i=1}^{N} x_i$$

$$\hat{C}_1 = \frac{1}{N} \Sigma_{i=1}^{N} (x_n - \hat{m}_{1ML})(x_n - \hat{m}_{1ML})^T$$

Since $L=0$ is represented by a Gaussian Mixture Model (GMM) of 2 Gaussians, and we have no knowledge of which component produced any given sample, we will need to use a more sophisticated approach to estimation. We have chosen to implement the Expectation Maximization (EM) algorithm in order to estimate the parameters of $p(x|L=0)$. The EM algorithm for GMMs operates as follows:

First, we initialize parameters. For weights w_1 and w_2 we initially assume all components of the GMM are weighted equally. For means m_{01} and m_{02} we randomly select two data points from D_{train}. For covariance C_{01} and C_{02}, we split D_{train} into two subsets based on the shortest Euclidean distance between each sample and either initial mean, we then take the sample covariance of those subsets.

Then, we calculate an update (represented by $t+1$) to each of these parameters using the following equations (as derived in $L03b\_Bilmes\_Tutorial...1998.pdf$):

$$p(j|x_i, \Theta^t) = \frac{w_j^t g(x_i|\Theta_j^t)}{\Sigma_{k=1}^{2} w_k^t g(x_i|\Theta_k^t)} \; \forall l_0 \in \{1,2\} \; and \; \forall i \in \{1,...,N\}$$

$$w_j^{t+1} = \frac{1}{N} \Sigma_{i=1}^{N} p(j|x_i, \Theta^t) \; \forall j \in \{1,2\}$$

$$m_{0j}^{t+1} = \Sigma_{i=1}^{N} \frac{p(j|x_i, \Theta^t)}{\Sigma_{i=1}^{N} p(j|x_i, \Theta^t)} x_i \; \forall j \in \{1,2\}$$

$$C_{0j}^{t+1} = \Sigma_{i=1}^{N} \frac{p(j|x_i, \Theta^t)}{\Sigma_{i=1}^{N} p(j|x_i, \Theta^t)} (x_i - m_{0j}^{t+1})(x_i - m_{0j}^{t+1})^T \; \forall j \in \{1,2\}$$

We then repeat this process until convergence. As suggested in $G/Code/EMforGMM.m$ we check convergence by getting a summation of the absolute value of the amount of change in each parameter and comparing that to a specified δ. If this "total absolute change" is less than δ for a number of iterations, we consider the algorithm converged. For time considerations, we use a delta of 0.02 in the algorithm that we implement directly in *Python*,

as this implementation is somewhat slow. Later in the assignment we use the $SciKitLearn$ implementation of the EM algorithm (which is built with much faster C++ code) which has a default δ of 0.001.

Since the EM algorithm is only guaranteed to find a local optimizer, we repeat this process 10 times with different random initializations each time. After all 10 repetitions are complete, we choose the set of parameters with the best log likelihood across the entire set of training data.

Once the estimation is complete, we generate an ROC curve and empirically find $\gamma_{\hat{P}_e}$ and $\hat{P}_e$ using the methodology detailed in the previous section.

We complete this process 3 times, once with each set of training data D_{train}^{10000}, D_{train}^{1000}, and D_{train}^{100}. Results for these 3 cases can be seen in Figure 2, 3, and 4 respectively.

Unsurprisingly the general trend is that more samples leads to a more accurate model, and thus a lower percentage of error.

Of note, however is that the D_{train}^{10000} actually resulted in a slightly worse $\hat{P}_e$ than D_{train}^{1000}. One explanation is that the data drawn for D_{train}^{10000} could have been somewhat unusual. Another possible consideration is that this application of the EM algorithm is only estimating 13 parameters (one weight is discounted due to the constraint that $w1 + w2 = 1$. As established in lecture, general practice is that one should have more samples than 10 times the square of the number of samples one intends to estimate. 1000 samples times dimensionality of 2 is more than this rule of thumb; as such, the increase in training data set size from 1000 to 10000 may have a diminishing return on the accuracy of the estimate. Considering that the classifier built from D_{train}^{10000} and D_{train}^{1000} achieve a $\hat{P}_e$ only 0.03% worse than the classifier built with knowledge of the true pdf, this seems likely.

Part 3: Classification with Logistic-Generalized-Linear Functions

We repeat the analysis a third time using a Logistic-Generalized-Linear Functions as approximate models instead of a GMM. First we use a Logistic Linear Function (LLF). After we use a Logistic Quadratic Function (LQF). $L05a_MaxLikelihoodTraining...LogisticGeneralizedLinearFucntions$ is used extensively as a reference for this analysis.

In the LLF case, our model is such:

$$P(L = 1|x) = h(x, \theta) = \frac{1}{1 + e^{-w^T b(x)}}$$

$$P(L = 0|x) = 1 - h(x, \theta)$$

$$where: \quad b(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \quad and \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

After substitutions and simplifications, this results in:

$$\hat{\theta}_{ML} = argmin_\theta = \frac{-1}{N} \Sigma_{i=1}^{N} [l_n ln(h(x_n, \theta)) + (1 - ln) ln(1 - h(x_n)]$$

which we are able to compute with $scipy.optimize.minimize()$ in order to estimate w.

For classification, we can rearrange the LRT into a more appropriate form:

$$\frac{h(x,\theta)P(L=0)}{(1-h(x|\theta)P(L=1)} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{P(L=0)}{P(L=1)} = \gamma$$

Note, $P(L=0)$ and $P(L=1)$ are estimated using the same technique as in the previous section.

As we did in the previous part, we train this classifier once with each training set, and produce ROC curves, $\gamma_{\hat{P}_e}$, and $\hat{P}_e$ for each classifier. As you can see in Figures 5, 6, and 7, using an LLF to approximate this case is a terrible idea. You can imagine that any attempt to separate the center Gaussian of label 1 from either outer Gaussian results in nearly totally misclassifying the other outer Gaussian. Attempts to get both outer Gaussians of label 0 in one classification result in nearly totally misclassifying label 1. Yet, the classifier would often find the $\hat{P}_e$ would arise from simply deciding every sample was from label 0, due to the relatively height weight of $P(L=0)$ at 0.6.

For the LQF case, our model is practically identical, with the exception that:

$$b(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix} \quad and \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

As you can see, in Figures 8, 9, and 10, this approach to approximating the true model is actually somewhat reasonable. One can easily image choosing a parabola that encircles the center Gaussian of label 1 without encroaching too much on either outer Gaussian of label 2.

Compared to the classifier built with an ML estimation of the GMM, or the classifier built with knowledge of the true PDF, $\hat{P}_e$ is reasonable. For D_{train}^{10000}, $\hat{P}_e$ is 17.66% - only about half a percent point worse than the true classifier or the estimated GMM classifiers with sufficient data. Decreasing samples to D_{train}^{1000} had minimal effect on $\hat{P}_e$. Decreasing samples further to D_{train}^{100} began to have a noticeable impact, with a jump in $\hat{P}_e$ of about one fifth of a percent point. Again, it seems that D_{train}^{1000} is of sufficient size to properly estimate the parameters. In this case, there are only 6 parameters being estimated - our rule of thumb would suggest that even 500 samples may be sufficient. Though also, we must remember that no amount of training data will overcome the inherent "lossiness" of such an approximation.

Overall, LQF is not a bad approximation *in this case*. In a case with more labels, or even just a more complicated arrangement of Gaussians, one can easily imagine scenarios where LQF would falter (for example if label 1 was more fully "boxed in"). Further, an ML estimation of GMMs with the EM algorithm does not seem so difficult that it needs to be approximated. However, though one can imagine a case where the true PDF is a GMM with absurdly many components that are arranged conveniently in a way that would allow you to approximate the decision boundary effectively with some parabola. In such a case, the massive reduction in number of parameters to be estimated (and thus number of samples necessary) may be helpful. Ultimately, it is clear that understanding the nature of your data is an incredibly important component of choosing a model properly.

Question 2

As per the assignment:

X is a two-dimensional random vector distributed according to a GMM with C components, equal weights, mean vectors spaced evenly on a line, and distinct circularly symmetric covariance matrices ($\Sigma_i = \lambda_i I$). We choose $C = 10$, $\mu_i = [i\ i]^T$. We choose by drawing C random random samples from a random variable uniformly distributed across $[0.2, 0.3]$.

For this assignment, we use Bayesian Information Criterion (BIC) and K-fold cross-validation (K-fold) to select the model order for an ML estimation of the true PDF. In this GMM case, "model order" refers to the number of components.

For illustrative purposes, we run the BIC and K-fold model order selection algorithms on training data sets drawn from the true PDF of size 10^p for $p \in \{2, 3, 4, 5\}$. For each of these p values, we run 100 experiments, each with an independently drawn training data set. Due to time constraints, we were unable to run experiments for $p = 6$.

In the BIC case: For each possible model order $M \in \{1, ..., 20\}$ we use the EM algorithm discussed previously (now generalized) to estimate a GMM PDF of M components $p_M(x|\theta_M)$ with parameters θ_M, and then we calculate a BIC value according to:

$$BIC = -2ln(p_M(D|\theta_M)) + n_M ln(N)$$

where $p_M(D|\theta_M)$ is the likelihood of the entire training data set under the estimated model, n_M is the number of unconstrained parameters, and N is the number of samples in the training data set. The derivation of this equation can be found in $L04a_Stoica...2004.pdf$. At the conclusion of the algorithm, we choose the model order $\hat{M}$ which minimizes this equation.

In th K-fold case: For each possible model order $M \in \{1, ..., 20\}$ we break the training data set into $K = 5$ parts (chosen for computation time considerations); for each $K \in \{1, ..., 5\}$ we set $D_{train} = D - D_K$ and $D_{val} = D_K$, use the EM algorithm trained on D_{train} to estimate a GMM PDF of M components $p_{MK}(x|\theta_{MK})$, and we evaluate the log likelihood of the entire validation set D_{val} under model MK; before moving to the next model order, we compute an average log likelihood across all K for the current model order. At the conclusion of the algorithm, we choose the model order $\hat{M}$ which maximizes the average loge likelihood.

The results of these experiments (including minimums, medians, maximums) can be seen in Figures 11 and 12.

It seems that the K-fold algorithm begins predicting the model order correctly at a lower orders of magnitude of N that BIC does. The penalty term in the BIC equation - as well as the derivation of BIC itself - assumes you have sufficiently many samples to eclipse the number of parameters being estimated. There seems to be more variance in model order selection for K-fold before it hits a "critical point" where it begins overwhelmingly predicting the true model order - for more complicated models, this variant period is could much longer. It may also be that larger values of K could reduce this variance, though we would need more time to verify. Speaking of time, it should be noted that K-fold was obviously *much* slower that BIC.

Due to the time/computation cost of K-fold, BIC seems to be best choice for very large sample sizes ($p \geq 5$ in this case). For smaller sample sizes, K-fold chooses more accurate

model orders than BIC, and the smaller sample size should diminish the time/computation cost of K-fold. For very small sample sizes ($p \leq 2$ in this case), it would seem that there is not enough data to make a meaningful determination about about the model without some preexisting knowledge of the model.

Appendix A: Code

https://github.com/EAGrimaldi/EECE5644_Assigment_2

Appendix B: Figures

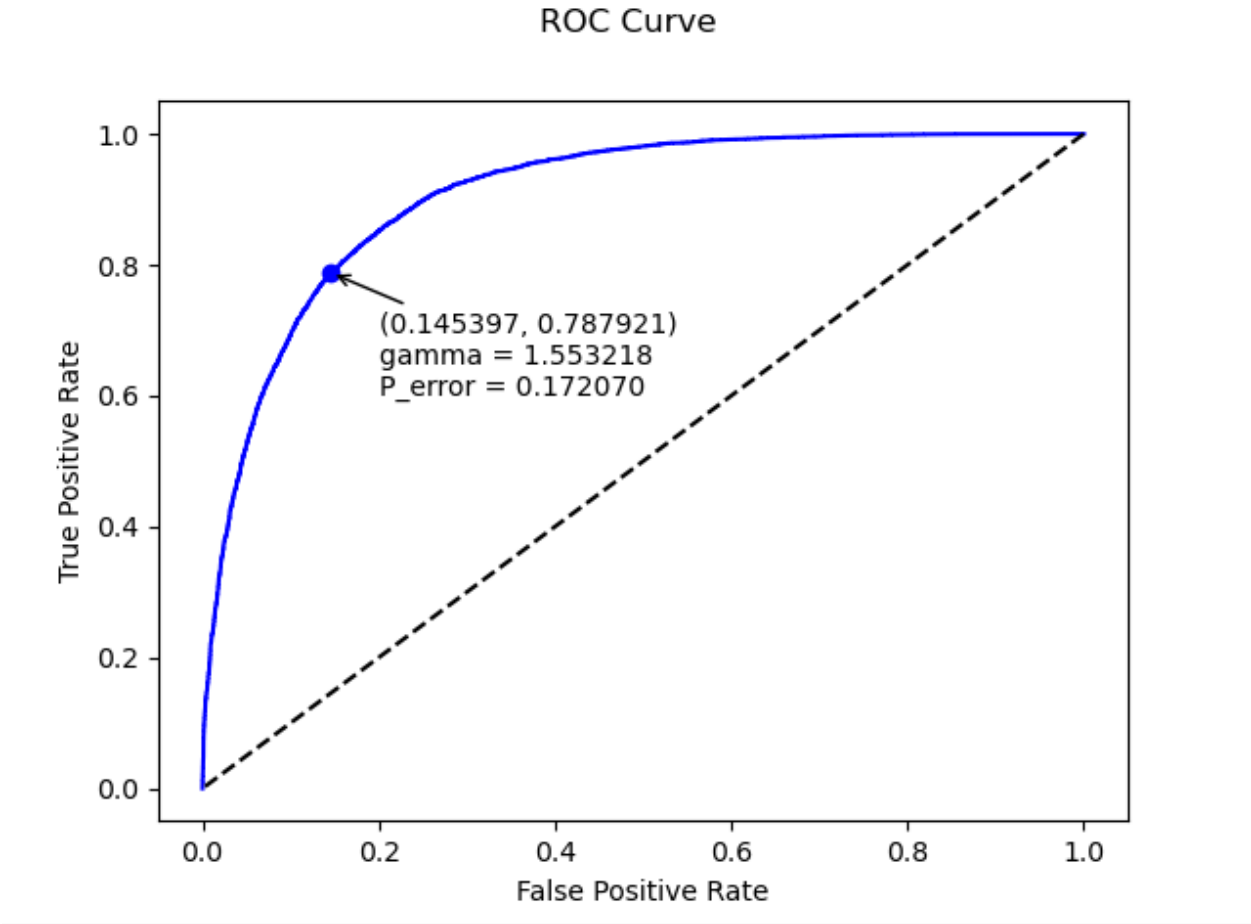

Figure 1: ROC curve for classifier from true PDF

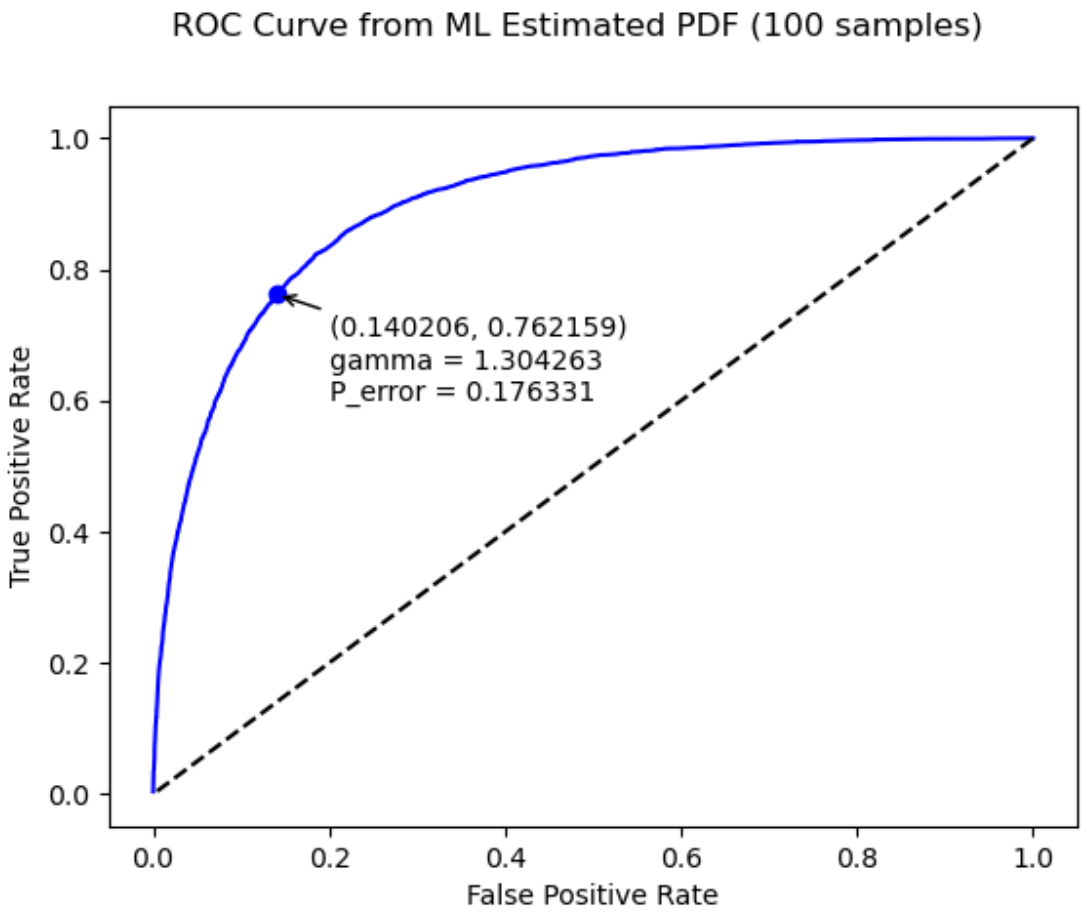

Figure 2: ROC curve for classifier from ML estimate of PDF with 100 samples

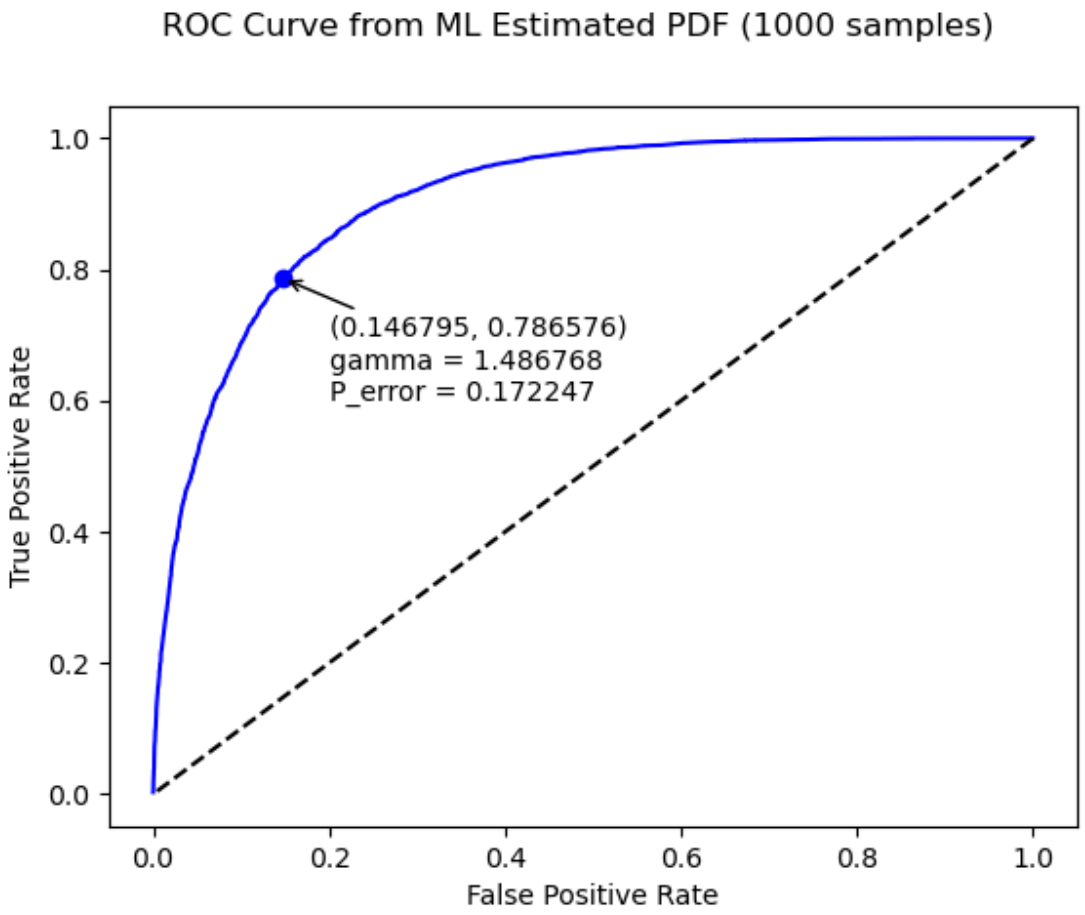

Figure 3: ROC curve for classifier from ML estimate of PDF with 1000 samples

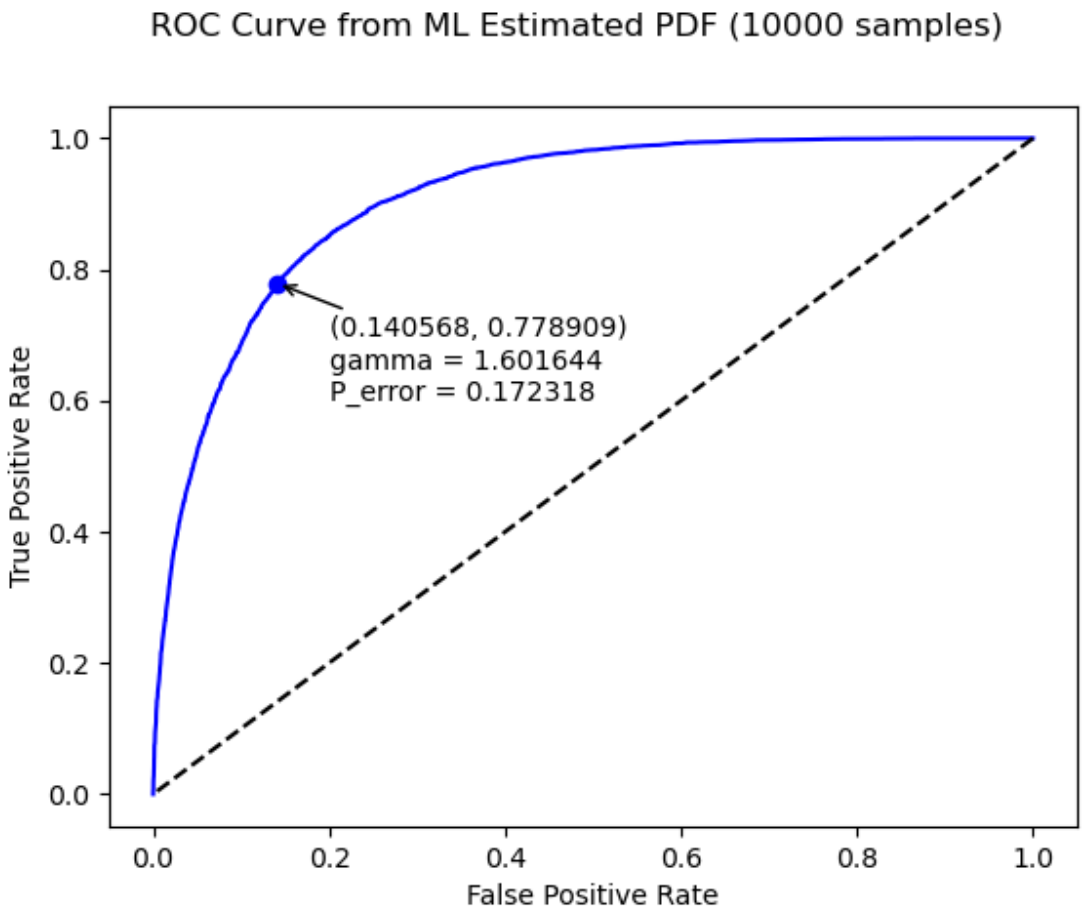

Figure 4: ROC curve for classifier from ML estimate of PDF with 10000 samples

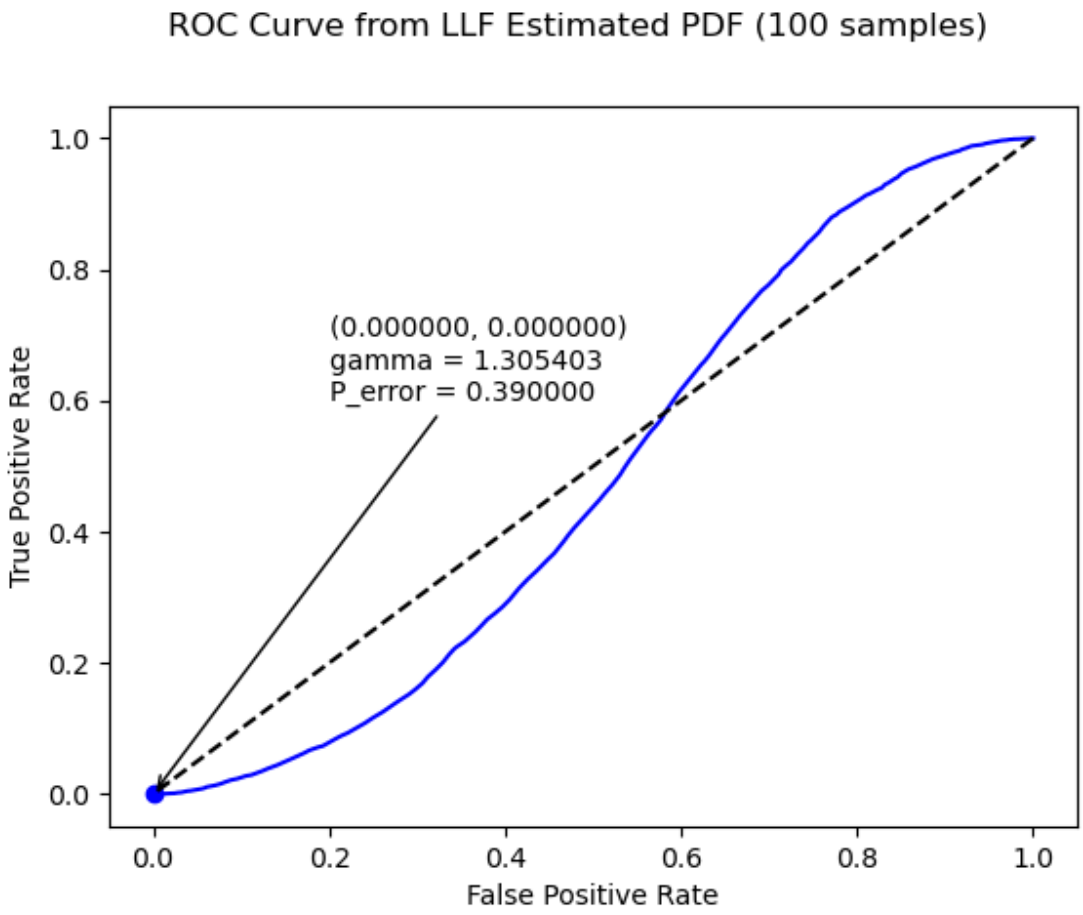

Figure 5: ROC curve for classifier from LLF approximation with 100 samples

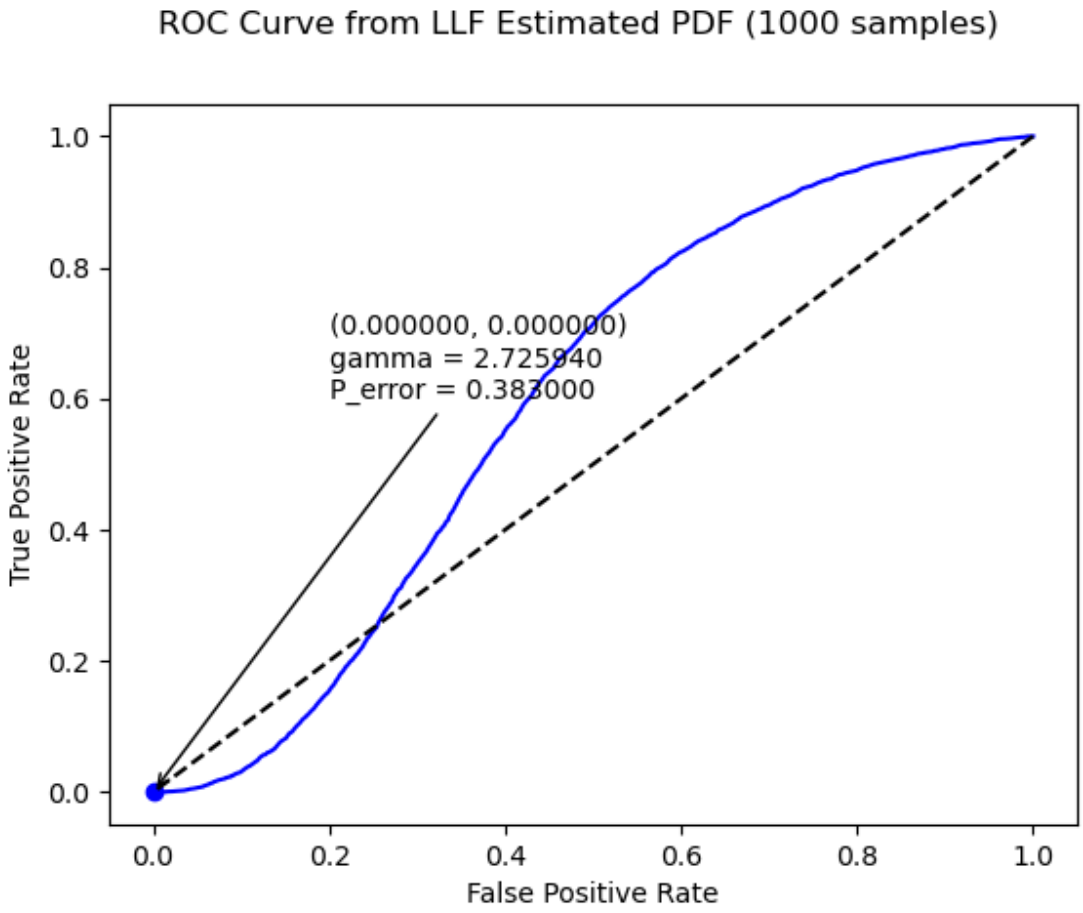

Figure 6: ROC curve for classifier from LLF approximation with 1000 samples

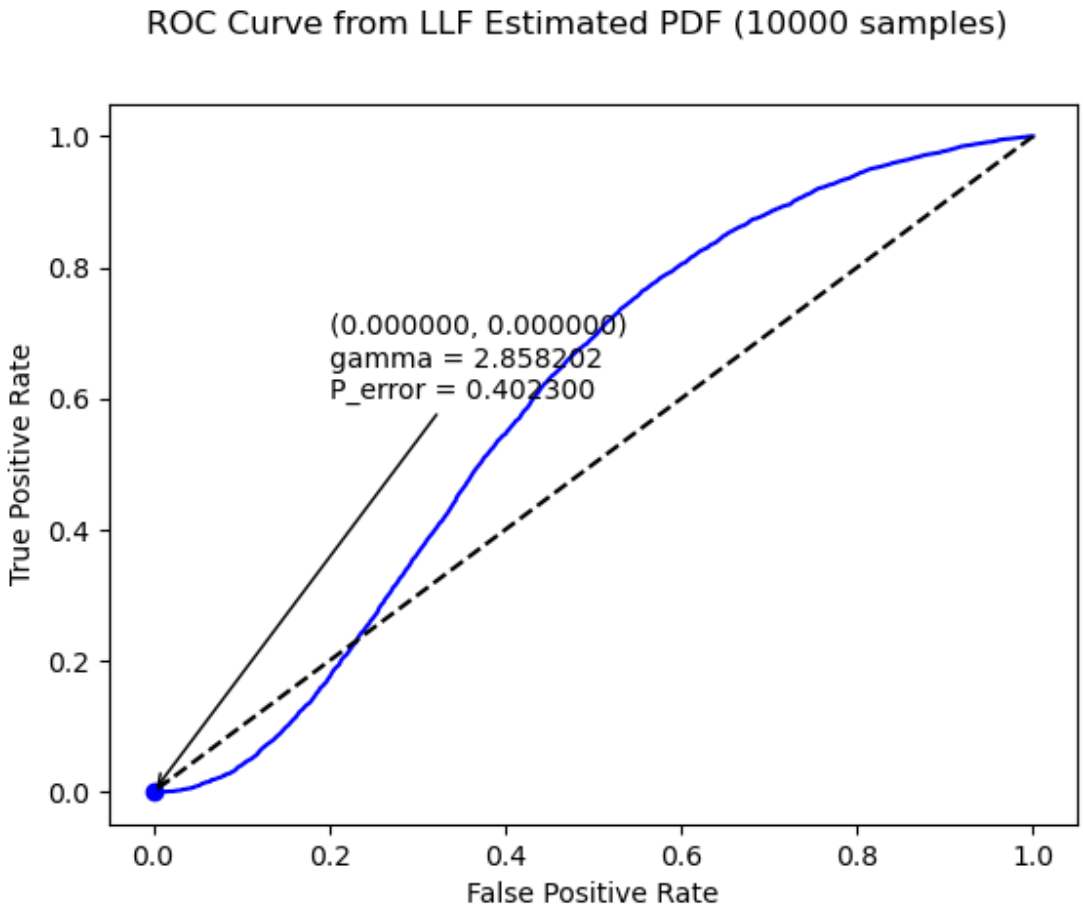

Figure 7: ROC curve for classifier from LLF approximation with 10000 samples

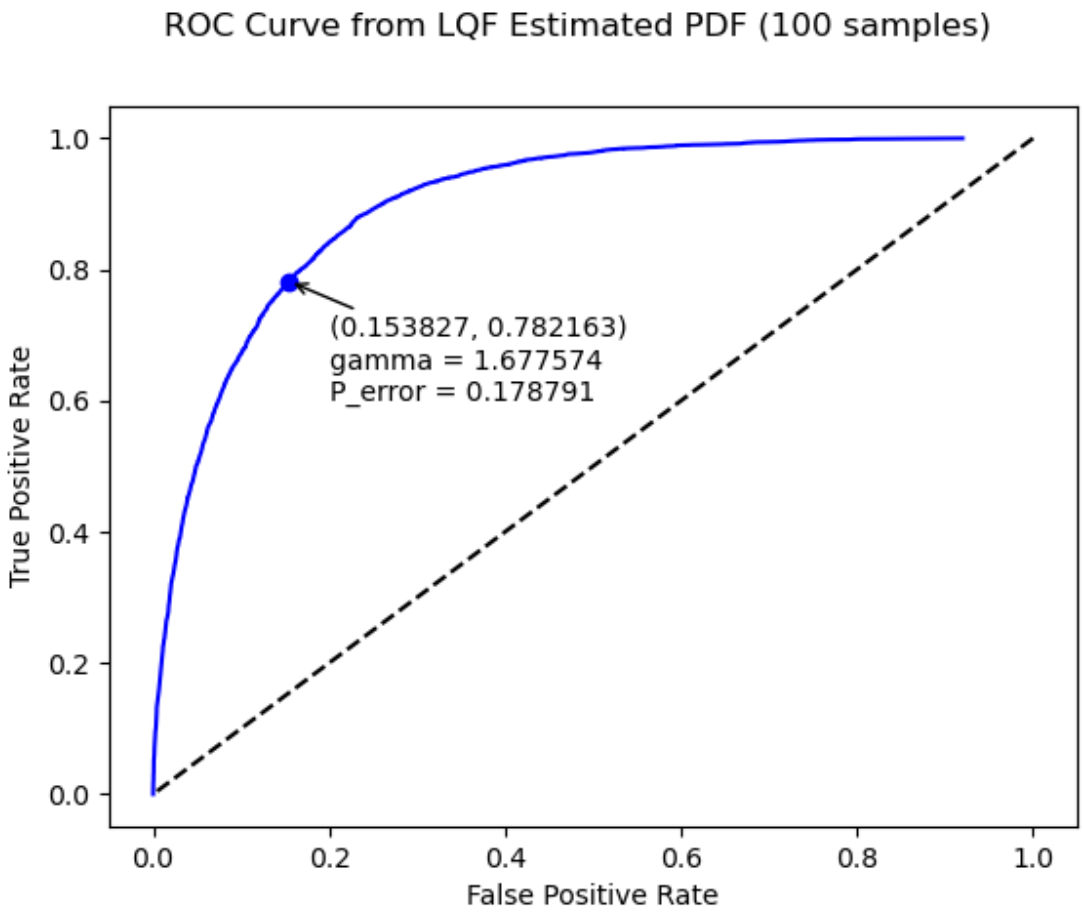

Figure 8: ROC curve for classifier from LQF approximation with 100 samples

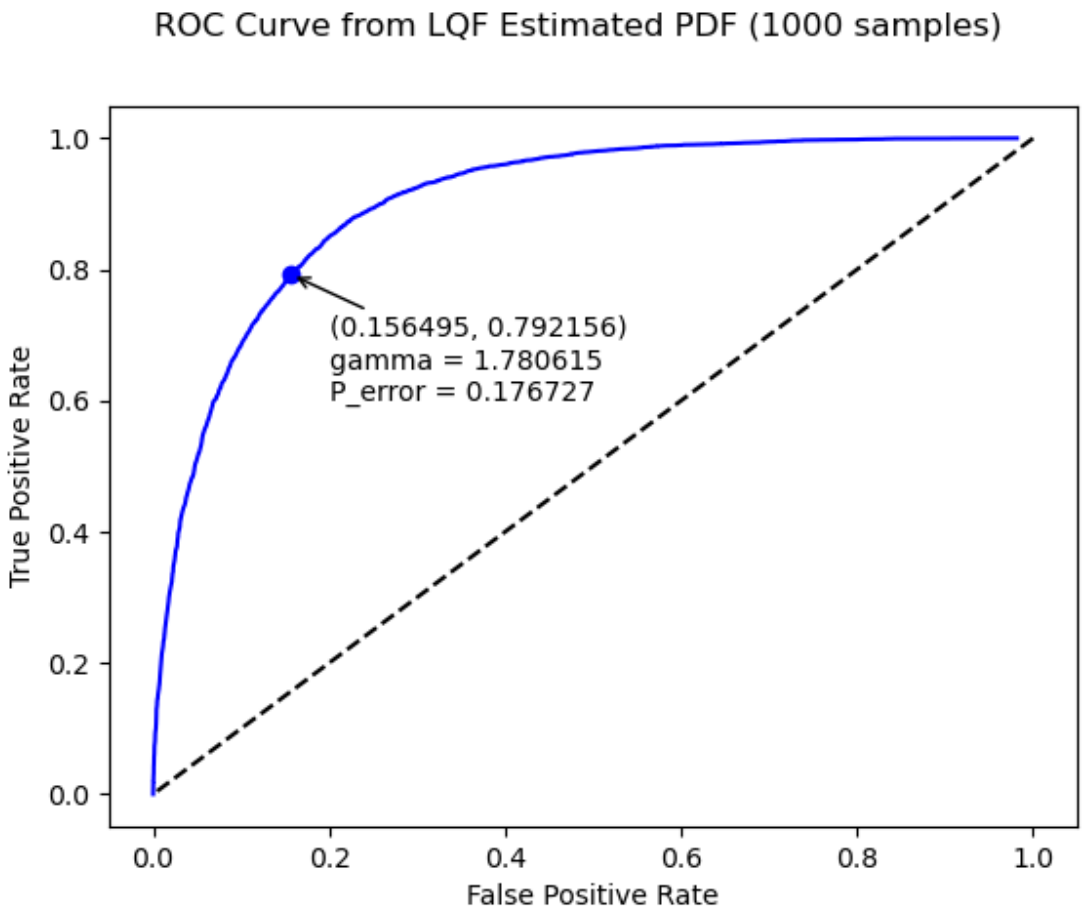

Figure 9: ROC curve for classifier from LQF approximation with 1000 samples

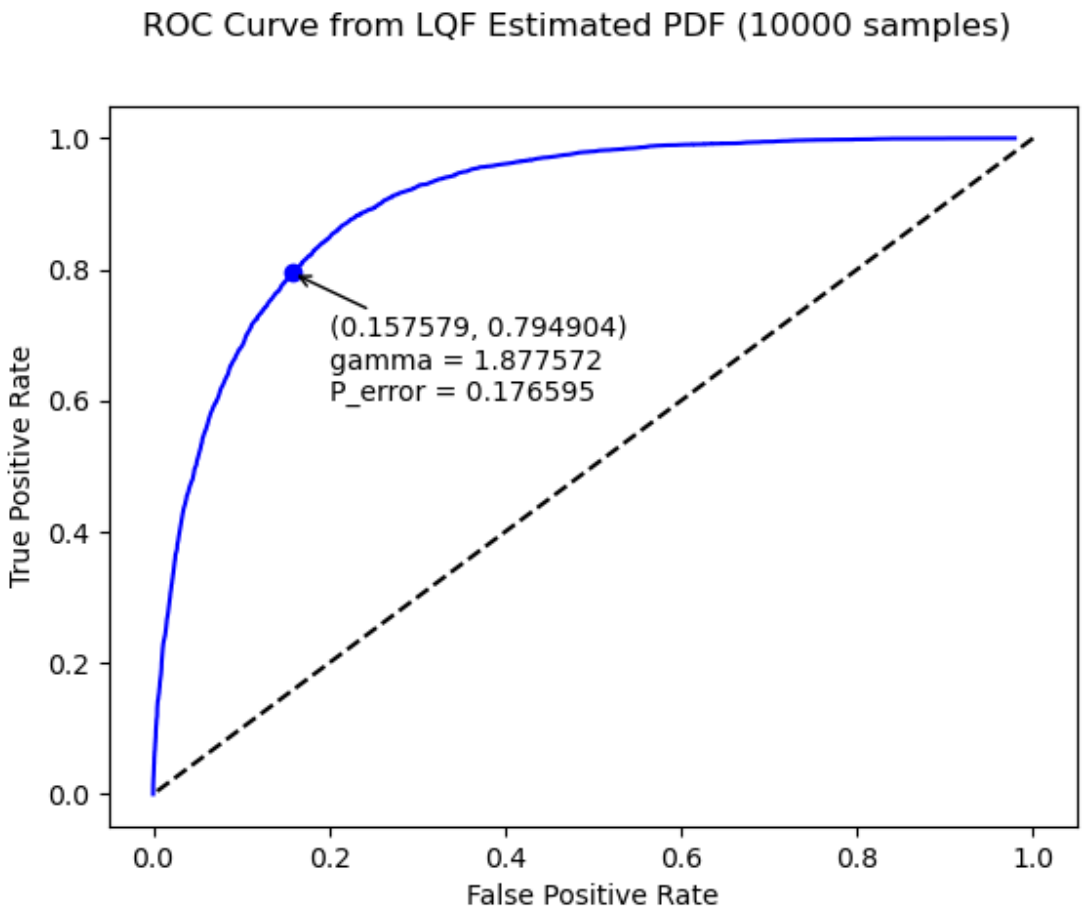

Figure 10: ROC curve for classifier from LQF approximation with 10000 samples

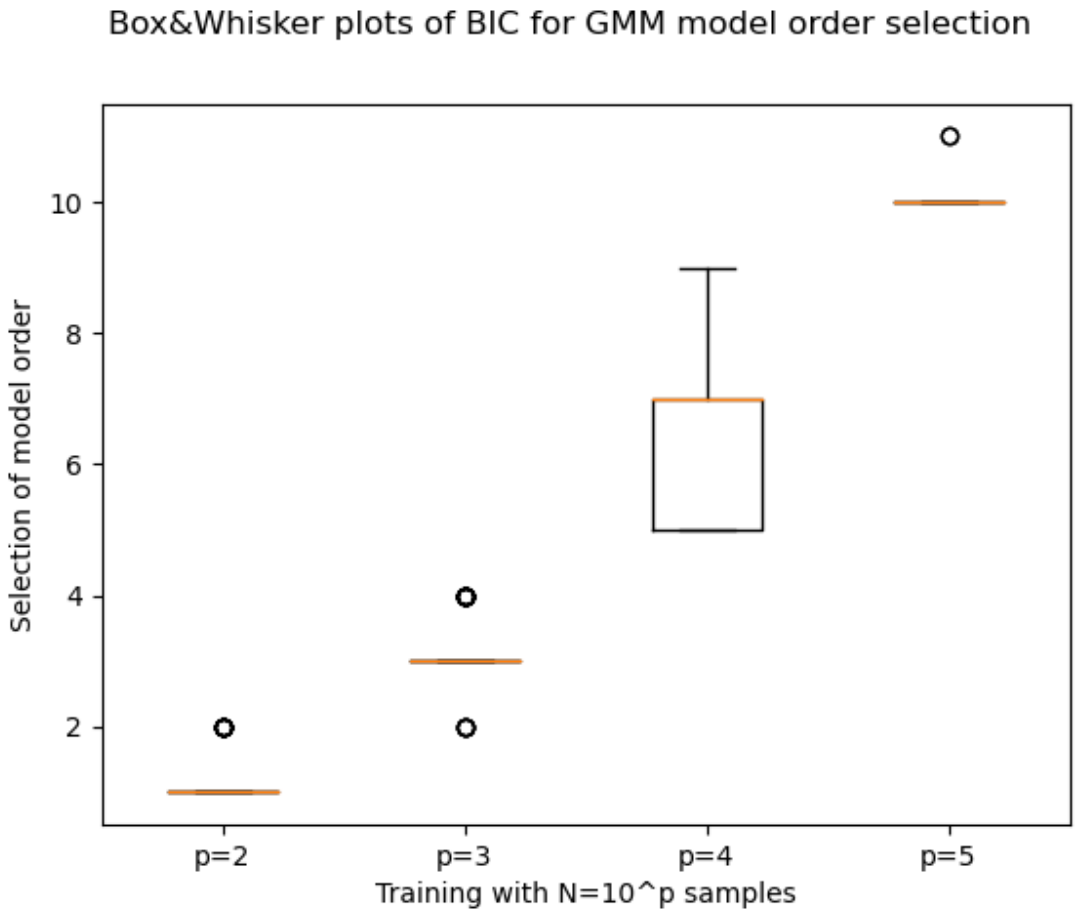

Figure 11: Box and whisker plot of model order selections with BIC across 100 experiments with training data sets of size $N = 10^p$

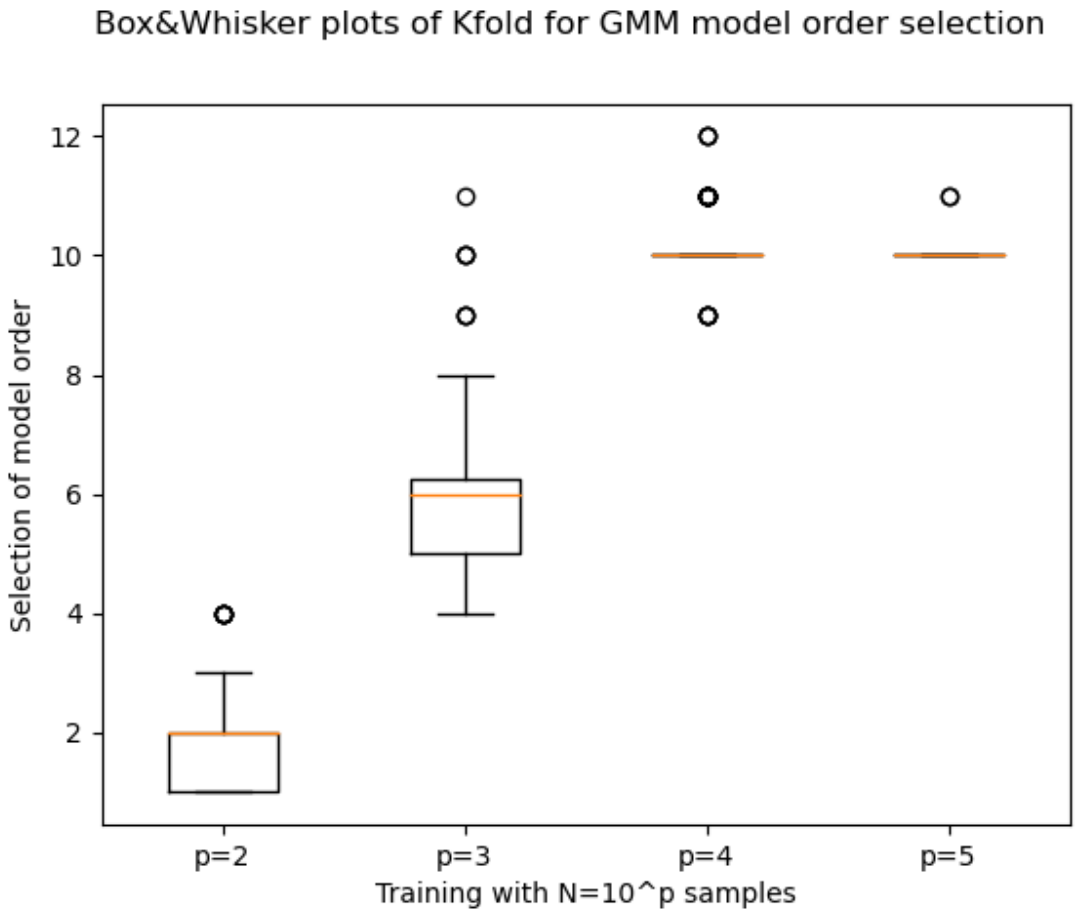

Figure 12: Box and whisker plot of model order selections with K-fold across 100 experiments with training data sets of size $N = 10^p$