Pattern Recognition

Exercise 2

Report Delivery Date: 9 May 2018

Student: Konidaris Vissarion 2011030123

Software: Matlab code

---

1. In this exercise we estimate the number of learnable parameters/weights of an artificial neural network. The input layer accepts data of size $3 \times 63 \times 64 = 12288$. The first hidden layer has 100 hidden units, followed by a second hidden layer with the same number of units/neurons. Finally the output layer has 10 outputs, one for each class.

   In the first hidden layer, each neuron needs to have 12288 parameters, because this is the size of the input layer, and also a bias term. Thus the first hidden layer has $100 \times 12288 + 100 = 1228900$ parameters. Following the same logic, the second hidden layer has $100 \times 1228900 + 100 = 122890100$ parameters, and the output layer has $10 \times 122890100 + 10 = 1228901010$ parameters. Adding all those up we conclude that the ANN has a total number of 1353020010 learnable parameters.

2. Suppose we have n i.i.d samples $D = \{x_1, ..., x_n\}$ that are generated from a Poisson distribution that is parameterized by λ.

$$p(x|\lambda)\frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, 2, ... \quad , \quad \lambda > 0$$

   We will attend to find the MLE of parameters λ. The likelihood function is

$$L_D(\lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}.$$

The log-likelihood then is

$$l_D(\lambda) = log(\prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}) = \sum_{i=1}^{n} log(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}) =$$

$$= \sum_{i=1}^{n} (-\lambda + x_i log\lambda - logx_i!) = -n\lambda - \sum_{i=1}^{n} logx_i! + log\lambda \sum_{i=1}^{n} x_i.$$

The Maximum Likelihood Estimation of λ can be found by solving the convex optimization problem $\text{argmax}_\lambda(l_D(\lambda))$. (This optimization problem is convex because we are maximizing a concave function).

$$\nabla_{\lambda_{MLE}} l_D(\lambda) = 0 \Leftrightarrow -n + \frac{1}{\lambda_{MLE}} \sum_{i=1}^{n} x_i = 0$$

$$\Leftrightarrow \lambda_{MLE} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Therefore, λ_{MLE} is just the sample mean of the n observations in the sample.

3. In this exercise we implement a Maximum Likelihood - Naive Bayes Classifier in Matlab. We have 10 classes corresponding to digits from 0 to 9. Each sample is represented by a $28 \times 28 = 784$ binary image. In our Bayes classifier, we assume that the features, in our case the pixels, are independent. We will use a Bernoulli distribution to model each pixel of each class. Let x^{y_i} be the random variable that corresponds to the i^{th} pixel/feature of the digit y. Now we derive the $p^{y_i}_{MLE}$ of the Bernoulli distribution generating the i^{th} pixel of the a digit y.

$$L_{D_{x^{y_i}}} = P(D_{x^{y_i}} | p^{y_i}) = \prod_{j=1}^{n} P(X = x_j^{y_i}) = (p^{y_i})^k (1 - p^{y_i})^{k'}$$

where $k = \sum_{j=1}^{n} x_j^{y_i}$ and $k' = n - k$.

$$l_{D_{x^{y_i}}}(p^{y_i}) = log((p^{y_i})^k (1 - p^{y_i})^{k'}) = klog(p^{y_i}) + k' log(1 - p^{y_i})$$

$$\nabla_{p^{y_i}_{MLE}} l_{D_{x^{y_i}}}(p^{y_i}) = 0 \Leftrightarrow \frac{k}{p^{y_i}_{MLE}} - \frac{k'}{1 - p^{y_i}_{MLE}} = 0$$

$$\frac{k - kp_{MLE}^{y_i} - k'p_{MLE}^{y_i}}{p_{MLE}^{y_i}(1 - p_{MLE}^{y_i})} = 0 \Leftrightarrow p_{MLE}^{y_i} = \frac{k}{k + k'}$$

$$p_{MLE}^{y_i} = \frac{1}{n}\sum_{j=1}^{n} x_j^{y_i}$$

Or else the $p_{MLE}^{y_i}$ Bernoulli parameter of feature i of class y is given by the sample mean of this pixel of this class. The Matlab implementation of the Bayes classifier is in the file $exercise2\_3$ under the name $Exe3.m$. The code calculates the accuracy of the model and prints the learnable parameters of each class as well as the confusion matrix.

4. Assume that we are having an experiment where we take n consecutive measurement at times k=1,2,...,n. The data time series then is $H(n) = \{x_k, \quad k = 1, .., n\}$, where n=25. Suppose that the probability density function of the measurements is Gaussian with $p(x) \sim N(\mu, \sigma^2)$, with standard deviation $\sigma = 1.25$. Suppose also that the prior knowledge for the distribution of the mean μ is that $P(\mu) \sim N(\mu_0, \sigma_0^2)$. Using Bayesian estimation we know that

$$P(\Phi|H(n)) = \frac{1}{\sqrt{2\pi}\tau} exp[\frac{-1}{2\tau^2}(\Phi - \rho)^2]$$

$$p = \frac{\sigma^2\mu_0 + n\sigma_0^2\overline{x_n}}{\sigma^2 + n\sigma_0^2}$$

$$\tau^2 = \frac{\sigma^2\sigma_0^2}{\sigma^2 + n\sigma_0^2}$$

The Matlab code in the file $exercise2\_4$ with the name $Exe4.m$ prints $P(\Phi|H(n))$ in the same plot as n changes from 1 to 25, for $\mu_0 = 0$ and $\sigma_0^2 = 10\sigma^2$. The same is done for $\sigma_0^2 = \sigma^2$, $\sigma_0^2 = 0.1\sigma^2$ and $\sigma_0^2 = 0.01\sigma^2$. It is evident by the plots that when σ_0^2 is big(prior knowledge), the estimation of the mean converges faster to the sample mean of the time series measurements. When the σ_0^2 is small, the influence of the sample mean on the estimate of the mean is less, making the estimate to stay close to $\mu_0 = 0$.

5. In the final exercise we had to implement an experiment to conclude at which step in the process of a classification problem it is best to apply feature selection. We assume that our data are EEG measurements of 25 patients, where fifteen of them have been diagnosed with autism. Then we extract 1000 features from those measurements to use for learning. In reality we simulate this experiment by randomly generating 1000 features for each of the 25 patients. We then run three distinct classification processes. First we fit the data using an SVM classifier using the leave-one-out technique. In the second process we apply feature selection, using a similarity measure algorithm based on Pearson's correlation coefficient, in each iteration of the leave-one-out technique before fitting these 24 data points to the SVM. Finally, in the third classification process we apply feature selection only one time at the beginning on all the data before applying the fitting with the leave-out-out technique. The Matlab implementation is in the file $exercise2\_5$ under the name $overfittingTest.m$. The first two techniques are proven to be quite similar, achieving accuracy close to 50%. This accuracy is reasonable, as the random generated features do not provide us with any meaningful information. The third technique though overfits the data, as it achieves 100% accuracy.