CSE847 HW1

Javen W. Zamojcin
zamojci1@msu.edu

2024, September 21

1 Question 1

1.1 Part A

Prove $E[\hat{\sigma}^2_{MLE}]$!= σ^2

1. $E[\hat{\sigma}^2_{MLE}] = E[\frac{1}{N}\Sigma^N_{i=1}(x_i - \hat{\mu}_{MLE})^2]$
2. $E[\hat{\sigma}^2_{MLE}] = \frac{1}{N}E[\Sigma^N x_i^2 - 2\Sigma^N x_i\hat{\mu} + \Sigma^N\hat{\mu}^2]$
3. $E[\hat{\sigma}^2_{MLE}] = \frac{1}{N}E[\Sigma^N x_i^2 - 2N\hat{\mu}^2 + N\hat{\mu}^2]$
4. $E[\hat{\sigma}^2_{MLE}] = \frac{1}{N}E[\Sigma^N x_i^2 - N\hat{\mu}^2]$
5. $E[\hat{\sigma}^2_{MLE}] = \frac{1}{N}E[\Sigma^N x_i^2] - E[\hat{\mu}^2]$
6. $E[\hat{\sigma}^2_{MLE}] = E[x^2] - E[\hat{\mu}^2]$
7. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 + E[x_i]^2 - \hat{\sigma}^2_{MLE} - E[x_i]^2$
8. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 - \hat{\sigma}^2_{MLE}$
9. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 - Var(\hat{\mu}) = \sigma^2 - Var(\frac{1}{N}\Sigma^N x_i)$
10. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 - (\frac{1}{N})^2 * Var(\Sigma^N x_i)$
11. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 - (\frac{1}{N})^2 * N\sigma^2$
12. $E[\hat{\sigma}^2_{MLE}] = \sigma^2 - (\frac{1}{N}) * \sigma^2$
13. $E[\hat{\sigma}^2_{MLE}] = (\frac{N-1}{N}) * \sigma^2$
14. $E[\hat{\sigma}^2_{MLE}]$!= $\sigma^2 \implies Bias$

1.2 Part B

$\hat{\mu}_{MAP} = \frac{\frac{1}{\sigma^2}\Sigma_{i=1}^{N} x_i + \frac{\theta}{\lambda}}{\frac{N}{\sigma^2} + \frac{1}{\lambda}}$

2 Question 2

2.1 Part A

1. Likelihood Function

$= L(\lambda; k_1, ..., k_N)$

$= \prod_{i=1}^{N} (\frac{\lambda^{k_i} e^{-\lambda}}{k_i!})$

The events are independent implies the likelihood function is the product of all PMFs.

2. Log Likelihood Function

$= ln(\prod_{i=1}^{N} (\frac{\lambda^{k_i} e^{-\lambda}}{k_i!}))$

$= \sum ln(\frac{\lambda^{k_i} e^{-\lambda}}{k_i!})$

$= \sum [ln(e^{-\lambda}) - ln(k_i!) + ln(\lambda^{k_i})]$

$= \sum [-\lambda - ln(k_i!) + k_i * ln(\lambda)]$

$= -N * \lambda - \sum (ln(k_i!)) + ln(\lambda) * \sum (k_i)$

3. $\hat{\lambda}_{MLE}$

$= \frac{d}{d\lambda} L(\lambda; k_1, ..., k_N) = 0$

$= \frac{d}{d\lambda} (-N * \lambda - \sum (ln(k_i!)) + ln(\lambda) * \sum (k_i))$

$= -N + \frac{1}{\lambda} \sum (k_i)$

$\longrightarrow \lambda = \frac{1}{N} \sum (k_i)$

4. $\hat{\lambda}_{MLE}$ = sample mean of N observations $= \frac{1}{N} \sum_{i=1}^{N} (k_i)$

2.2 Part B

1. E[X]

$= \sum_{x \in X} x * P(x|\lambda)$

$= \sum_{x=0}^{\inf} x * (\frac{\lambda^x e^{-\lambda}}{x!})$

$= e^{-\lambda} \sum_{x=1}^{\inf} \frac{x}{x!} \lambda^x$

$= \lambda e^{-\lambda} \sum_{x=1}^{\inf} \frac{\lambda^{x-1}}{(x-1)!} \longrightarrow y = x - 1$

$= \lambda e^{-\lambda} \sum_{y=0}^{\inf} \frac{\lambda^y}{y!}$

$= \lambda e^{-\lambda} e^{\lambda}$

$= \frac{\lambda e^{\lambda}}{e^{\lambda}}$

$= \lambda$

3 Question 3

3.1 Part A

Number of Independent Parameters to Estimate

$= (n_{classes} - 1) + (n_{attributes} * n_{parameters})$

$= (2 - 1) + (2 * 3)$

$= 7$

3.2 Part B

$P(Y = yes) = \frac{4}{10}$, $Count(Y = yes) = 4$, $Count(Y = no) = 6$

$P(Size = small|Y = yes) = \frac{1}{4}, P(Size = large|Y = yes) = \frac{3}{4}$

$P(Size = small|Y = no) = \frac{3}{6}, P(Size = large|Y = no) = \frac{3}{6}$

$P(Color = green|Y = yes) = \frac{0}{4}, P(Color = red|Y = yes) = \frac{4}{4}$

$P(Color = green|Y = no) = \frac{5}{6}, P(Color = red|Y = no) = \frac{1}{6}$

$P(Shape = Irr|Y = yes) = \frac{1}{4}, P(Shape = Circ|Y = yes) = \frac{3}{4}$

$P(Shape = Irr|Y = no) = \frac{4}{6}, P(Shape = Circ|Y = no) = \frac{2}{6}$

3.3 Part C

$x = (size = small, color = red, shape = circ)$

$P(y = no|x) = P(size = small|y = no) * P(color = red|y = no) * P(shape = circ|y = no) * P(y = no)$

$P(y = no|x) \propto \frac{3}{6} * \frac{1}{6} * \frac{2}{6} * \frac{6}{10} = 0.0167$

$P(y = yes|x) \propto \frac{1}{4} * 1 * \frac{3}{4} * \frac{4}{10} = 0.075$

$P(y = no|x) = \frac{P(y=no|x)}{P(y=no|x)+P(y=yes|x)} = \frac{0.0167}{0.0167+0.075} = 0.182$

$P(y = yes|x) = \frac{P(y=yes|x)}{P(y=yes|x)+P(y=no|x)} = \frac{0.075}{0.0167+0.075} = 0.8179$

$P(y = yes|x) > P(y = no|x) \longrightarrow y* = yes$

4 Question 4

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \; y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; \lambda = 0.001$$

1. $w_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$

$\longrightarrow z = w_0 * X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

$\longrightarrow \hat{y} = \sigma(z) = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$

$\longrightarrow \Delta = avg(y - \hat{y}) * X^T = \begin{bmatrix} 0.0 & 0.167 & 0.167 & 0.167 & 0.167 & 0.167 \end{bmatrix}$

$\longrightarrow w_1 = w_0 + 0.001(\Delta) \longrightarrow ...$

$\longrightarrow w_{100} = \begin{bmatrix} 0.0 & 0.0165981 & 0.0165981 & 0.0165981 \end{bmatrix}$, $LLH = -4.109$

2. $w_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$

$\longrightarrow ... \longrightarrow w_{100} = \begin{bmatrix} 0.0 & 0.0165981 & 1.0089 & 0.0165981 \end{bmatrix}$, $LLH = -3.362$

The log likelihood loss function is both convex and differentiable, meaning that no saddle-points exist and only a single (global) minimum exists, implying that only a single optimal weight solution exists to converge to. So in theory, given

enough iterations and a small enough learning rate, the "final" converged weight vectors should approximately be the same regardless of the initial values.

However, in practice my results were not as expected. After 100 iterations of my algorithm implementation, the final weight vectors and loss values were significantly different. This may be because my implementation used a constant number of iterations and didn't incorporate the epsilon convergence condition, or possibly my learning rate wasn't small enough and the global optima was overshot.

5 Question 5

5.1 Part A

5.1.1 i.)

How many independent parameters are there in this Gaussian Naive Bayes classifier? What are they?

$n_{parameters} = 4m + 1 = 4d + 1 = \{P(Y = 1) = \theta, \mu_{11}, \sigma_{11}, ..., \mu_{dk}, \sigma_{dk}\}$

5.1.2 ii.)

Can we translate w into the parameters of an equivalent Gaussian Naive Bayes classifier without any extra assumption?

We can translate the learned weights of a LR classifier to the parameters of an equivalent GNB with the assumption of variance equivalence ($\sigma_{ik} = \sigma_i$).

$w_i = \sum_i (\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2})$

5.2 Part B

5.2.1 i.) Model Implementation Descriptions

Gaussian Naive Bayes:

My Gaussian Naive Bayes (GNB) implementation includes `fit()`, `predict()`, `probability-xy()`, and `probability-yx()` functions for discrimination.

`fit()` is responsible for grouping the training samples by class and calculating their respective standard deviations and means. It also calculates the class probabilities (`p(y)`) by dividing the respective class sample counts by the total sample count.

`predict()` accepts a test sample dataframe, calculates the respective class probabilities given a sample (`p(y-x)`), and assigns the most probable class to each

sample. `predict()` makes use of the `probability-yx()` function to achieve this.

`probability-yx()` calculates the probability of a class given a sample by first calculating the probability of a sample given a class (`p(x-y)`) through the `probability-xy()` function and multiplying this by the probability of the respective class (`p(y)`).

`probability-xy()` calculates the probability of a sample x, given a class y (`p(x-y)`). This is done through the Gaussian probability density function and taking the product of the partial feature probabilities.
$GaussianPDF(x, y) = (\frac{1}{\sigma_{xy} * \sqrt{2\pi}}) * \exp - \frac{(x-\mu_{xy})^2}{2*\sigma_{xy}^2}$

Logistic Regression:

My Logistic Regression (LR) model accepts parameters for learning rate (default `0.01`) and max iterations (default `1000`), but doesn't make use of an epsilon variable in this implementation. It has the functions `fit()`, `predict()`, `sigmoid()`, and `log-likelihood()` for training and discriminating.

`fit()` first initializes the weight vector to all zeros, matching the dimension of the number of features. It then performs gradient ascent for the number of max iterations: the linear hypothesis (z) as the dot product of the training samples and weight vector; the probability (y-hat) as the `sigmoid()` of the linear hypothesis; the gradient as the mean of the difference between the true (y-true) and predicted (y-hat) values of y, multiplied by the sample matrix; the weight update as the learning rate multiplied by the gradient; and the `log-likelihood()` of the y-true and y-hat, which is stored for later use.

The `sigmoid()` function simply calculates the output of the logistic curve in the domain of [0, 1].
$sigmoid(z) = \frac{1}{1+\exp -z}$

The `log-likelihood()` method accepts the parameters y-true (yt) and y-hat (yh) and calculates the log likelihood loss function. First, it makes use of the epsilon variable as a filler value for zero and one values in the given y-hat vector. Then it calculates the likelihood value.
$likelihood(yt, yh) = \sum(yt * \log(yh)) + (1 - yt) * \log(1 - yh)$

Finally, the `predict()` function accepts testing samples as a parameter and outputs their respective predicted discrete classes. This is done by calculating the probabilistic linear hypothesis (y-hat) again, and applying a binary threshold function where the class 1 is chosen if the probability is greater than 0.5, or class 0 otherwise.

5.2.2 ii.) Model Accuracy vs. Training Size

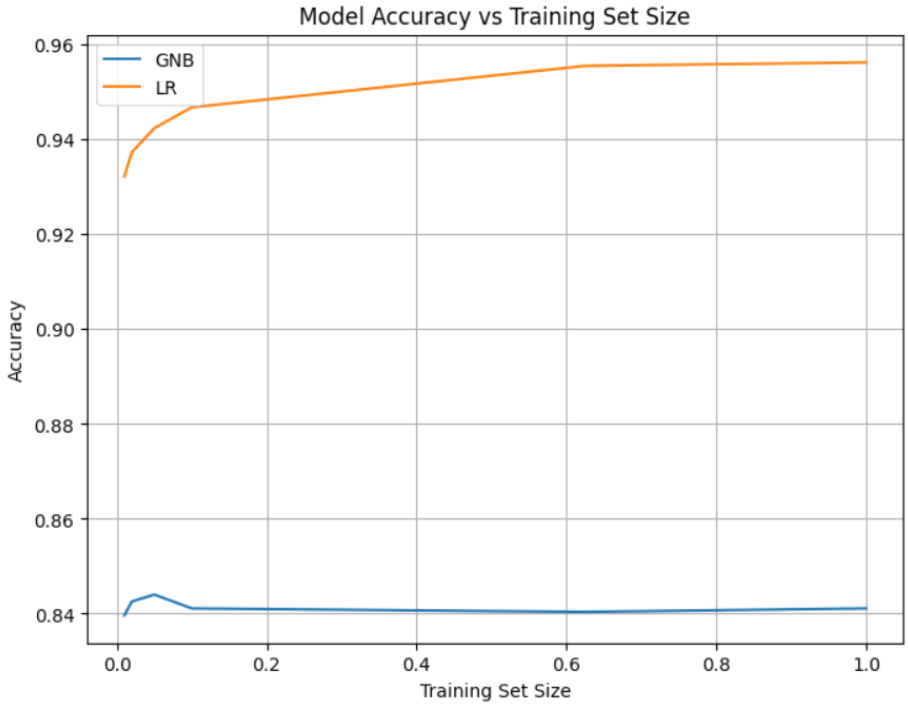

Figure 1: Q5 Model Accuracy vs Training Size

5.2.3 iii.) Sample Generation

k_idx	data	stat	variance	skewness	curtosis	entropy
0.0	generated	variance	3.886810	28.475792	24.506623	4.027396
		mean	-1.898866	-1.120628	1.726110	-1.593570
	train	variance	3.538848	29.212768	27.686654	4.288974
		mean	-1.868443	-0.993576	2.148271	-1.246641
1.0	generated	variance	3.491784	26.339279	27.118995	4.135338
		mean	-2.058375	-0.839185	1.842334	-1.248411
	train	variance	3.538848	29.212768	27.686654	4.288974
		mean	-1.868443	-0.993576	2.148271	-1.246641
2.0	generated	variance	3.330786	24.701685	29.032855	4.210647
		mean	-1.916735	-1.235472	1.731363	-1.120963
	train	variance	3.538848	29.212768	27.686654	4.288974
		mean	-1.868443	-0.993576	2.148271	-1.246641

Figure 2: Q5 Generated Sample Statistics

Figure (2) displays the metrics from generating 400 samples from my trained GNB models, one for each of the three folds, and comparing to their respective training fold data sets. You can see there is a non-trivial difference in both variance and mean between the generated and training data sets, for each of the folds. My guess for this phenomenon is because of the Gaussian assumption toward our data—that is, each class conditional feature is assumed to originate from an independent Gaussian distribution with its own mean and variance. It may leave room for error in our prediction of samples.