Project 1

A. Burak Gulhan

February 4, 2023

Contents

1 Part 1: Linear Regression

Write what you are planning on doing, a bit of information about each one, any troubles you had with the project. Make sure to cite anything. The following are some useful rules for publishing in CVPR. For this part of the project, I derived the closed form equations for ML and MAP, and compared and plotted the resulting predicted curves for different sample numbers (N) and model dimensions (M).

1.1 Derivation of Maximum Likelihood

Let X be a vandermonde matrix and let $x^{\circ k}$ represent element-wise power of vector x to the power of k.

Since the steps to obtain the error function was explained in detail in class, I am starting derivation from the error function.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 \tag{1}$$

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial w_k} &= \frac{1}{2} * 2\sum_{n=1}^{N}\left\{(y(x_n,\mathbf{w}) - t_n) * \frac{\partial}{\partial w_k}(y(x_n,\mathbf{w}) - t_n)\right\} && (2)\\
&= \sum_{n=1}^{N}\left\{(y(x_n,\mathbf{w}) - t_n)x_n^{\circ k}\right\} && (3)\\
&= (\mathbf{x}^{\circ k})^T(X\mathbf{w} - \mathbf{t}) \text{ remove sum and write as matrix and vectors} && (4)
\end{aligned}$$

Found result of taking derivative with respect to single element of w. Now take derivative with respect to vector w.

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= (\mathbf{x}^{\circ k})^T(X\mathbf{w} - \mathbf{t}) && (5)\\
&= \begin{bmatrix} —(\mathbf{x}^{\circ 0})^T— \\ —(\mathbf{x}^{\circ 1})^T— \\ \vdots \\ —(\mathbf{x}^{\circ})^T— \end{bmatrix}(X\mathbf{w} - \mathbf{t}) && (6)\\
&= X^T(X\mathbf{w} - \mathbf{t}) \text{ rewrite matrix as X transpose} && (7)
\end{aligned}$$

To minimize error function, set derivative equal to 0:

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= 0 && (8)\\
X^T(X\mathbf{w} - \mathbf{t}) &= 0 && (9)\\
X^TX\mathbf{w} - X^T\mathbf{t} &= 0 && (10)\\
X^TX\mathbf{w} &= X^T\mathbf{t} && (11)\\
\mathbf{w} &= (X^TX)^{-1}X^T\mathbf{t} && (12)
\end{aligned}$$

1.2 Derivation of Maximum Posterior

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2\beta}\mathbf{w}^T\mathbf{w} \tag{13}$$

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}(\frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2) + \frac{\partial}{\partial \mathbf{w}}(\frac{\alpha}{2\beta}\mathbf{w}^T\mathbf{w}) && (14)\\
&= X^T(X\mathbf{w} - \mathbf{t}) + \frac{\partial}{\partial \mathbf{w}}(\frac{\alpha}{2\beta}\mathbf{w}^T\mathbf{w}) \text{ calculated left derivative in equation 7} && (15)\\
&= X^T(X\mathbf{w} - \mathbf{t}) + \frac{\alpha}{\beta}\mathbf{w} && (16)\\
&= X^T(X\mathbf{w} - \mathbf{t}) + \lambda\mathbf{w} \text{ represent } \frac{\alpha}{\beta} \text{ as } \lambda && (17)
\end{aligned}$$

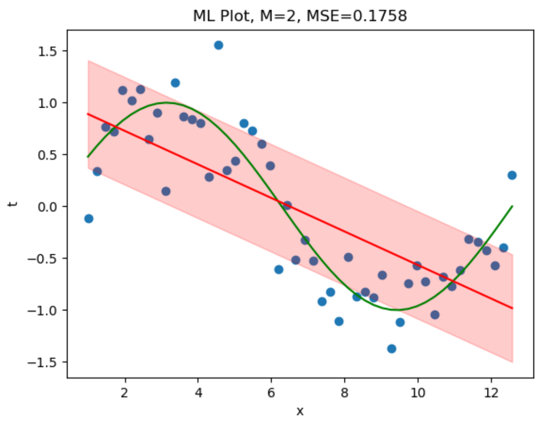
To minimize error function, set derivative equal to 0:

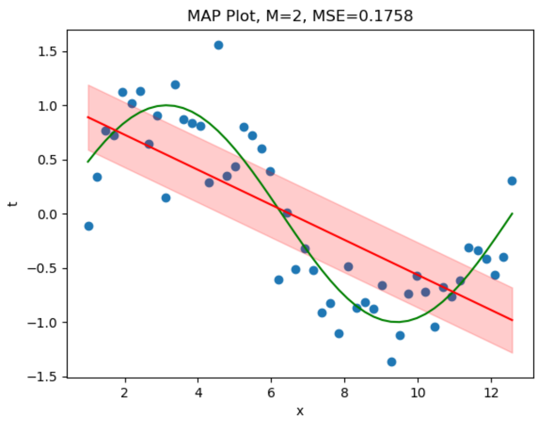
$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = 0 && (18)\\
X^T(X\mathbf{w} - \mathbf{t}) + \lambda\mathbf{w} = 0 && (19)\\
X^TX\mathbf{w} - X^T\mathbf{t} + \lambda\mathbf{w} = 0 && (20)\\
X^TX\mathbf{w} + \lambda\mathbf{w} = X^T\mathbf{t} && (21)\\
(X^TX + \lambda I)\mathbf{w} = X^T\mathbf{t} \text{ , I is the identity matrix} && (22)\\
\mathbf{w} = (X^TX + \lambda I)^{-1}X^T\mathbf{t} && (23)
\end{aligned}$$

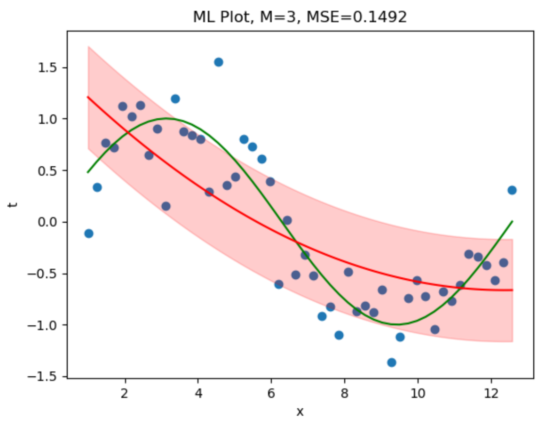
1.3 Visualization

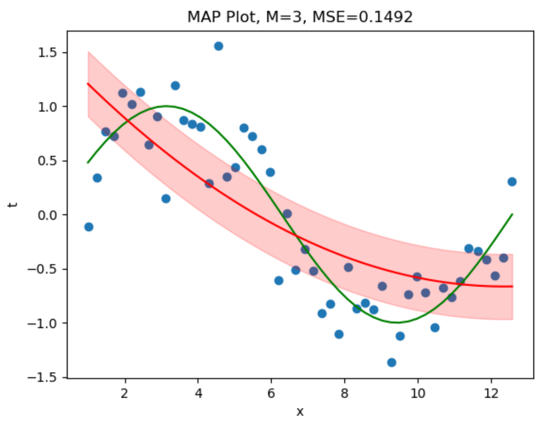
1.3.1 Varying M

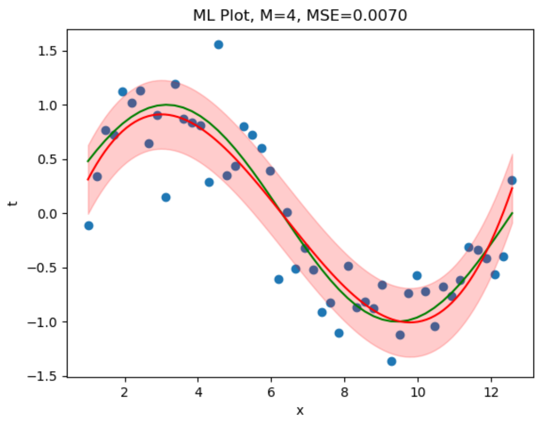
The following images are for N=50 training points and the order of the polynomial, M, is varied. The green line is the ground truth and the redi line is the predicted curve. The pink regions are one standard deviation of the predicted model. For MAP, the parameters are $\alpha = 0.05$ and $\beta = 11.1$
From observing the plots and Mean Squared Error (MSE), we see that the best performing M value is when M=4. This is expected since M=4 is the simplest model that can model the sine curve, thus minimizing overfitting. Another interesting observation is that when M≤4 the predicted models are the same and they have the same MSE, but when M¿4 we can see that the ML model has a higher MSE compared to MAP.

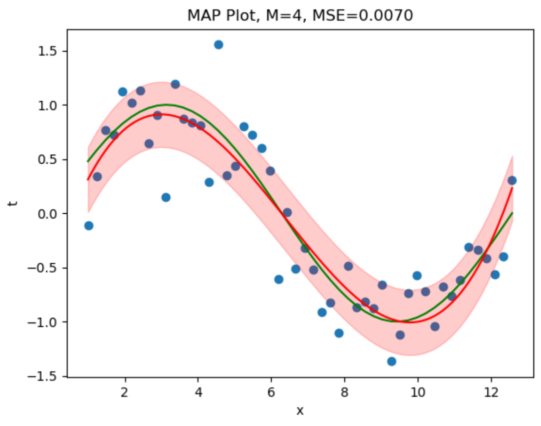

ML Plot, M=2, MSE=0.1758
x
t

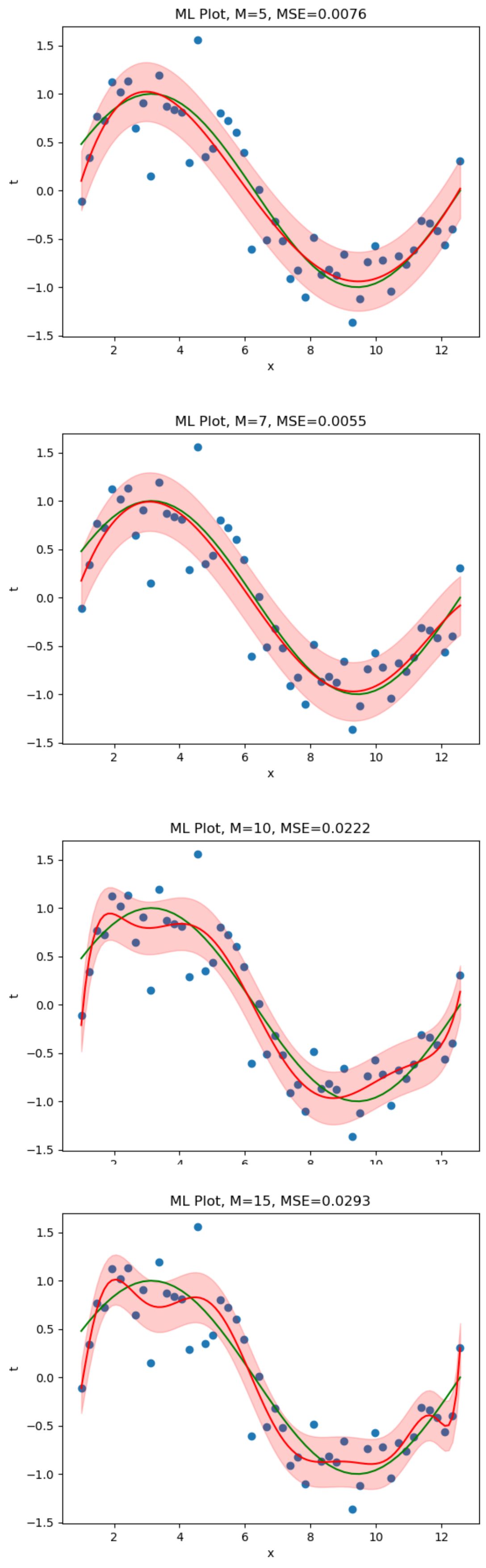

MAP Plot, M=2, MSE=0.1758
x
t

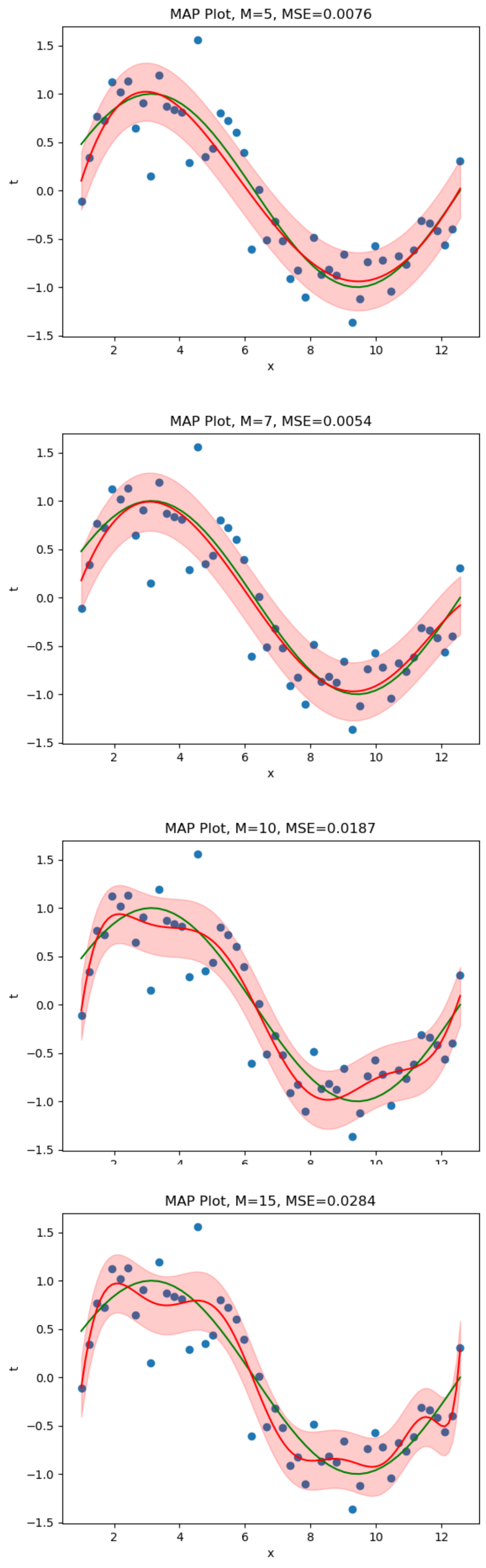

ML Plot, M=3, MSE=0.1492
x
t


MAP Plot, M=3, MSE=0.1492
x
t


ML Plot, M=4, MSE=0.0070
x
t


MAP Plot, M=4, MSE=0.0070
x
t

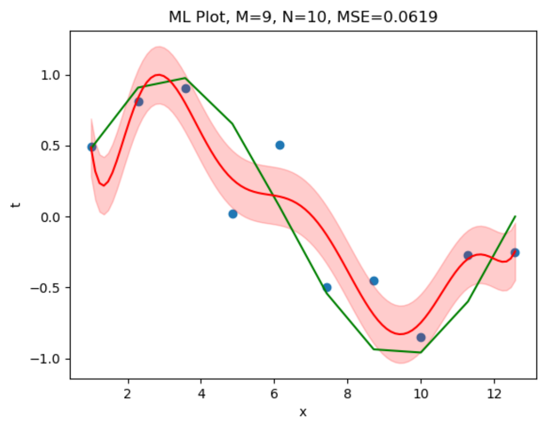

ML Plot, M=5, MSE=0.0076
ML Plot, M=7, MSE=0.0055
ML Plot, M=10, MSE=0.0222
ML Plot, M=15, MSE=0.0293
x
t
(a) ML

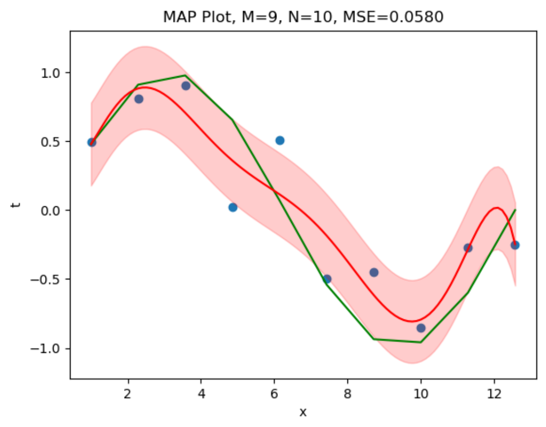

MAP Plot, M=5, MSE=0.0076
MAP Plot, M=7, MSE=0.0054
MAP Plot, M=10, MSE=0.0187
MAP Plot, M=15, MSE=0.0284
x
t
(b) MAP

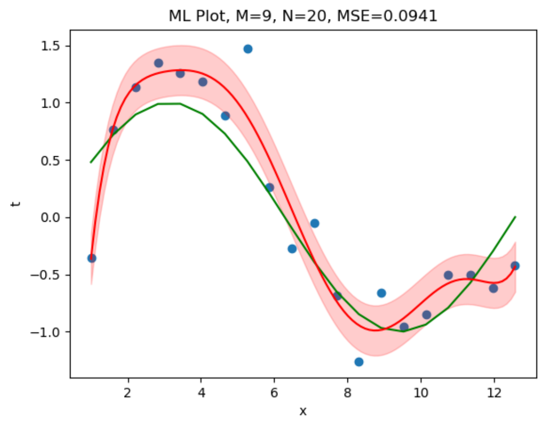
1.3.2 Varying N

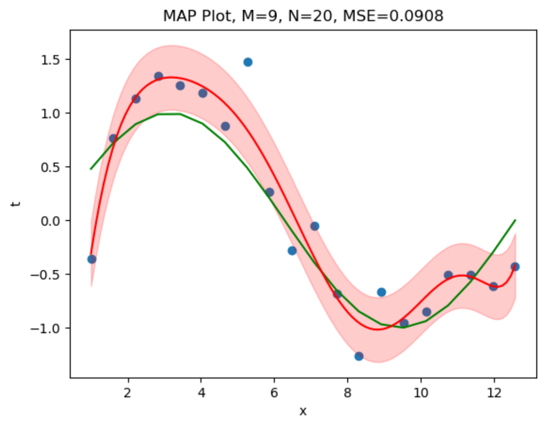
The following images show plots for M=9 and where N is varied.

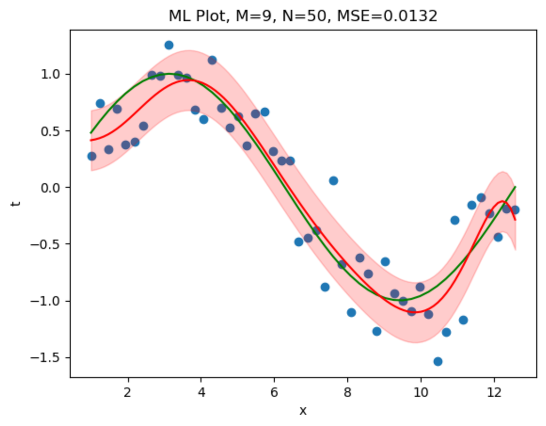
We can observe that as the number of points increases, the prediction becomes closer to the gorund truth and the MSE decreases, which is an expected result. When N is small we can see that there is a high amount of overfitting, but when N becomes larger the overfitting decreases, thus improving prediction. We can also observe that MAP is better or equal to ML in all these cases (for these results, $\alpha = 0.05$ and $\beta = 11.1$).

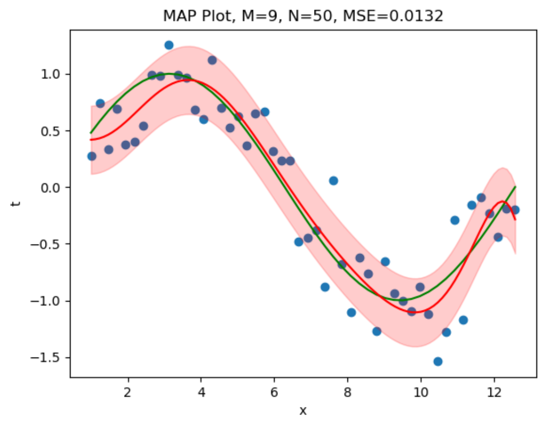


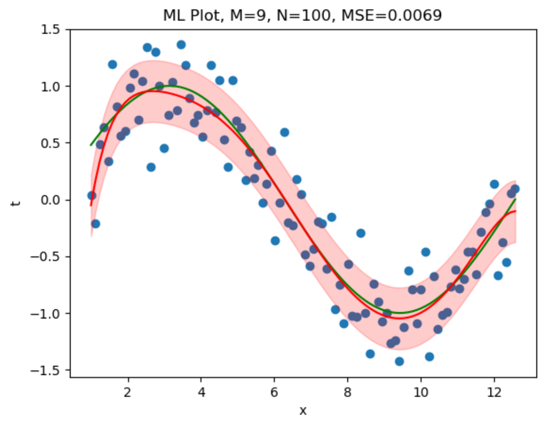




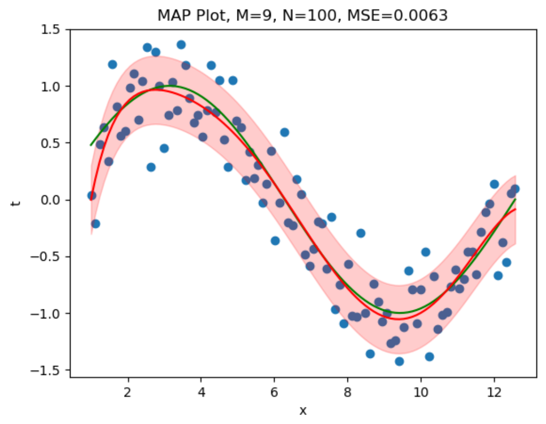








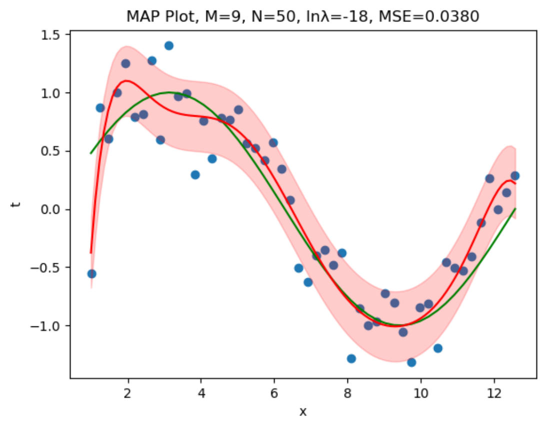
(a) ML

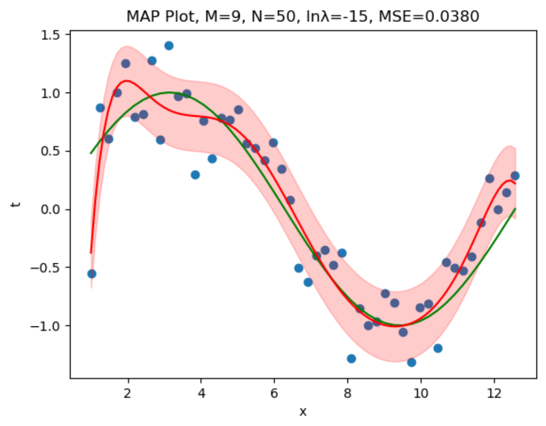


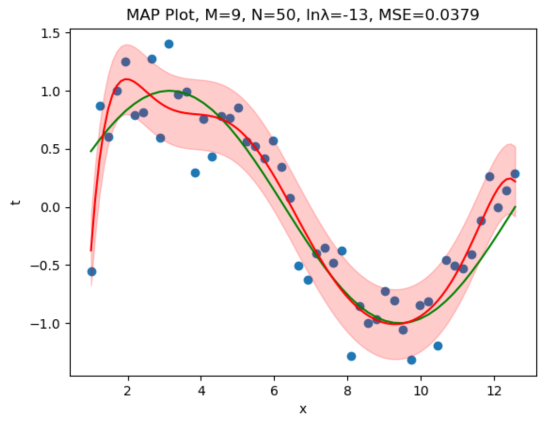
(b) MAP

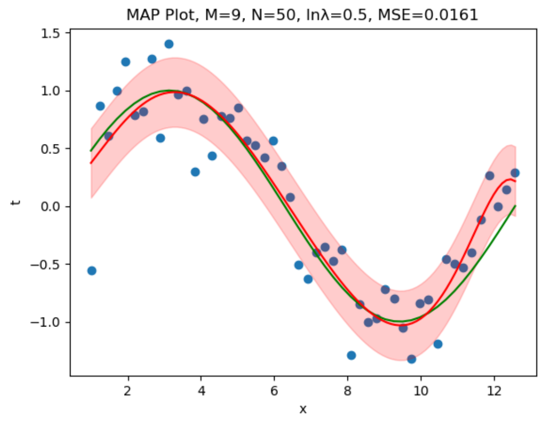
1.3.3 Varying λ

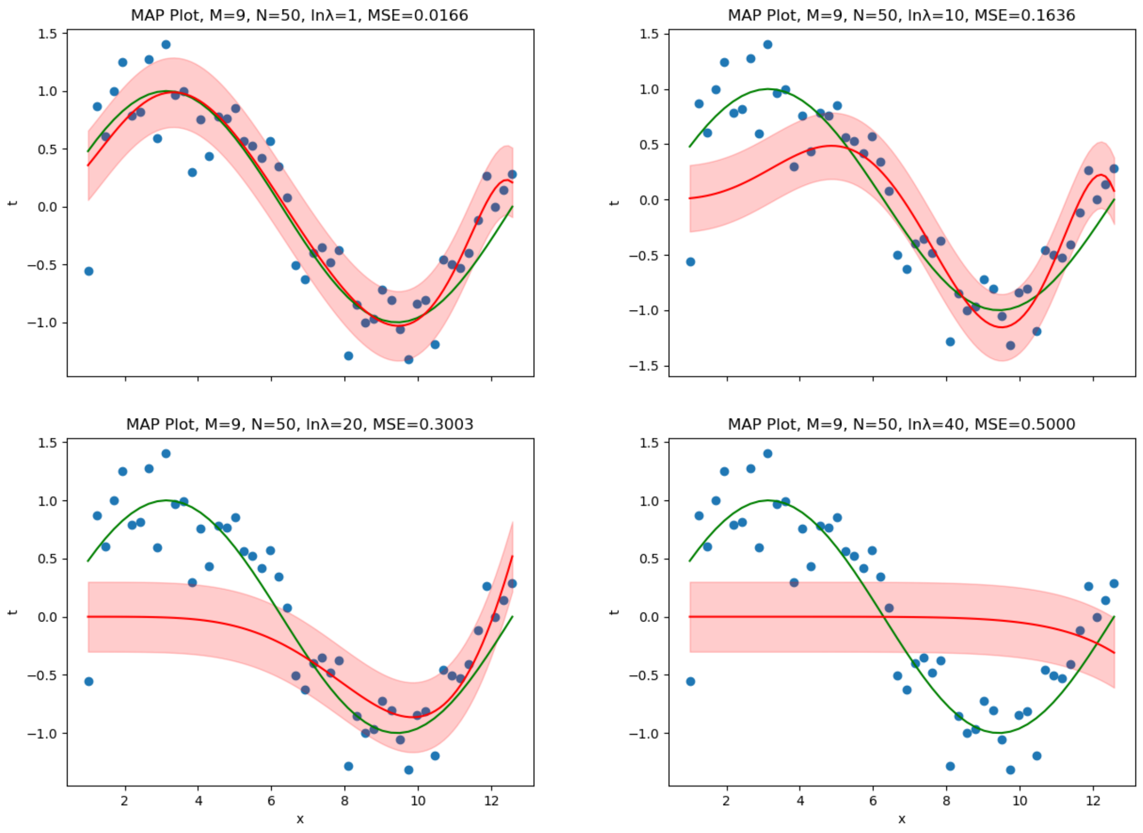
The following images show the MAP model with λ varied. We observe that for $\lambda = e^-18, e^-15, e^-13$ there is barely any change in the model, unlike the images in the textbook. Only when λ is large de we see visible changes in the plot. For the generated points for the figures, $\ln\lambda = 0.5$ gives the best results (lowest MSE) among other $\ln\lambda$ values. When λ becomes very large, the model becomes similar to a horizontal line, since model weights are penalized heavily in the error function and become very close to 0.











MAP Plot, M=9, N=50, lnλ=1, MSE=0.0166
MAP Plot, M=9, N=50, lnλ=10, MSE=0.1636
MAP Plot, M=9, N=50, lnλ=20, MSE=0.3003
MAP Plot, M=9, N=50, lnλ=40, MSE=0.5000
t
x

1.3.4 Investigating curse of dimensionality

From class we learned that due to the "curse of dimensionality" we need more sample points the higher dimensional model we have. From the figures obtained above, we can observe that increasing M for a fixed N gives increasingly worse results after a certain M value. Additionally, we observe that increasing N (number of samples) gives better results for a fixed M, even if M is large. So for small N and large M we see the largest effect (that is overfitting) from the curse of dimensionality. On the other hand, a small M (around 4) and large N, gives the best results.

1.4 Difficulties

A problem that I had in implementing ML and MAP in Python was that I was getting numerical errors when solving for w. when M $\geq$ 10, I was getting unexpectedly bad results, where the predicted curve looked similar to an exponential curve instead of a sine curve. The reason for this was that I was taking the inverse of X^TX which leads to numerical errors, especially when M is large. To solve this problem, instead of calculating W with the closed form solution, I solved $(X^TX)^{-1}X^T\mathbf{t}$ using a linear system of equations solver of the form Ax=b (where A $= X^TX$ $x = w$ and $b = X^Tt$) in Numpy (np.linalg.solve) which had much better numerical behavior.

2 Part 2: Classification

2.1 Introduction

Describe all the data you used using exact amounts. For later projects, how you extracted the features out of the data.

2.2 Derivation

Describe the methods you used in your project. For example, any derivations or proofs you have used.

Fisher's Criterion (for 2 classes):

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_1^2} \text{ where } s_k \text{ is the variance of class k and } m_k \text{ is the mean of class k} \tag{24}$$

or alternatively for any number of classes k:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \tag{25}$$

where

$$S_W = \sum_{k=1}^{K} S_k \text{ within class variance, where } S_K \text{ is the variance of class k} \tag{26}$$

and

$$S_B = \sum_{k=1}^{K} N_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \text{ between class covariance, where } m_k \text{ is the mean of class k} \tag{27}$$

an alternative way to calculate S_B, which I used in my code:

$$S_B = S_T - S_W \tag{28}$$

where

$$S_T = \sum_{n=1}^{N} (x_n - \mathbf{m})(x_n - \mathbf{m})^T \text{ total covariance matrix} \tag{29}$$

We want to choose a projection direction $\mathbf{w}$ such that the distance between class means is maximized and the inter-class variance is minimized. That is, we want to maximize $J(W)$.

Using S_W and S_B we can obtain the equation:

$$(\mathbf{w}^T S_B \mathbf{w}) S_W \mathbf{w} = (\mathbf{w}^T S_W \mathbf{w}) S_B \mathbf{w} \tag{30}$$

Since only the direction of $\mathbf{w}$ is important, we can ignore the scaling factors $(\mathbf{w}^T S_B \mathbf{w})$ and $(\mathbf{w}^T S_W \mathbf{w})$:

$$S_W \mathbf{w} \propto S_B \mathbf{w} \tag{31}$$

$$\mathbf{w} \propto (S_W^{-1} S_B) \mathbf{w} \tag{32}$$

$$\lambda \mathbf{w} = (S_W^{-1} S_B) \mathbf{w} \text{ we can add a constant before w and replace } \propto \text{ with } = \tag{33}$$

This is a standard eigenvalue problem and w can be solved using an eignenvalue solver.

2.3 Visuals and Results

From the results of plotting the data without any Fisher projection using KNN (figure 7 and figure 8), we see that very good results are obtained for train and test data with F1 scores 0.969 and 0.914 respectively. The highest misclassified class is class 0 (transitional pose) in both train and test, which corresponds to the transitions between Taiji poses. This might be due to transitional poses having a high amount of variance and being difficult to distinguish from other poses (this can be observed from the output of the example$_decision_boundary()function$).

For plotting with Fisher projection to $K - 1 = 7$ dimensions (as is recommended in the textbook), we obtain very bad results (figure 9 and figure 10). The training results have a F1 score of 0.956 and this not much worse compared to KNN without Fisher.

However, for test results we see that the prediction is much worse, with a F1 score of only 0.575. The large difference between train and test results is indicative of overfitting. It seems that most of the mispredictions – similar to using KNN without Fisher projection – are due to predicting class 0 (the transitional phase) as another class, or by predicting a different class as class 0. This is most likely due to Fisher projection removing information that would be helpful for classifying class 0. However, we can observe that some classes still have good results, such as class 6 and to a lesser extent class 3.

I also tried projecting onto different dimensions to see if they perform better. The best dimension I found for projecting was 41. The results for Fisher projecting onto 41 dimensions can be seen in figure 10 and figure 11. We can see that class 0 is still a problem, but in general all other classes have relatively good predictions. However, this is still much worse than using KNN with no projection (0.783 vs 0.914 F1 score).

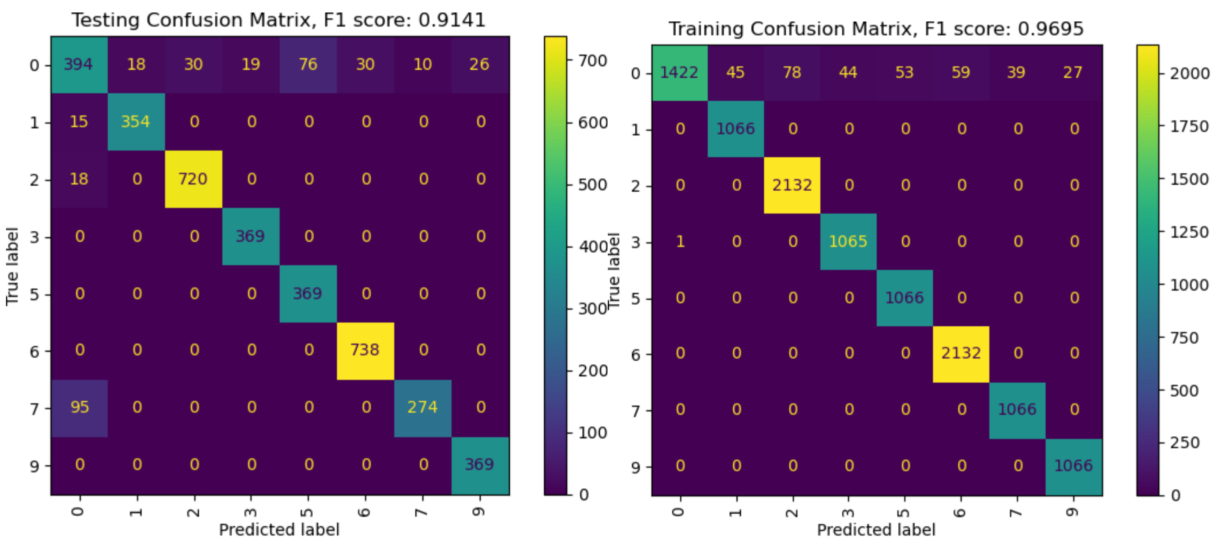


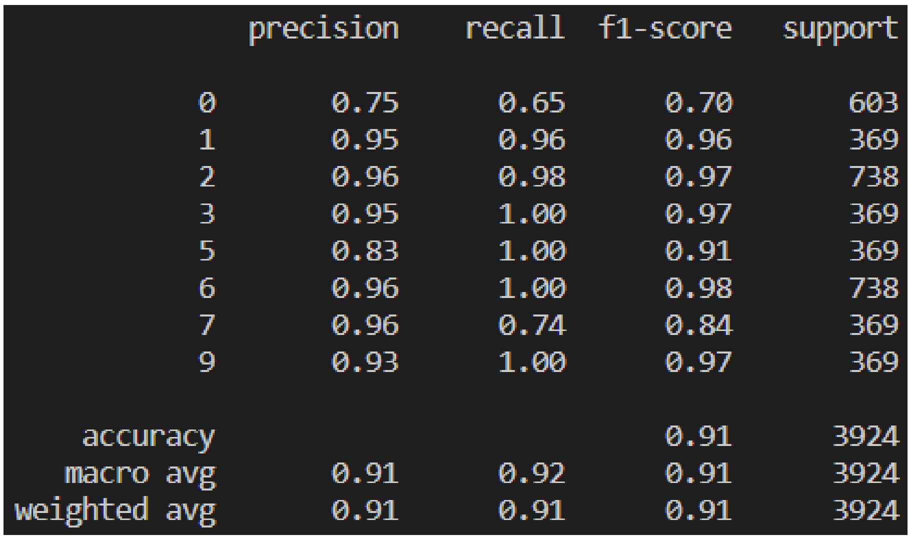
Figure 7

An example \subsubsection{} to organize your multiple methods.

Example Paragraph: You may also use \paragraph{} if you don't want it numbered and the text to start on the same line.

2.4 Difficulties

One problem I had with this part of the project was that the eigenvalue solver for Numpy returned complex valued vectors. To deal with this problem, I reduced the precision of matrix S_B to 3 decimal places, which eliminated any complex valued results.

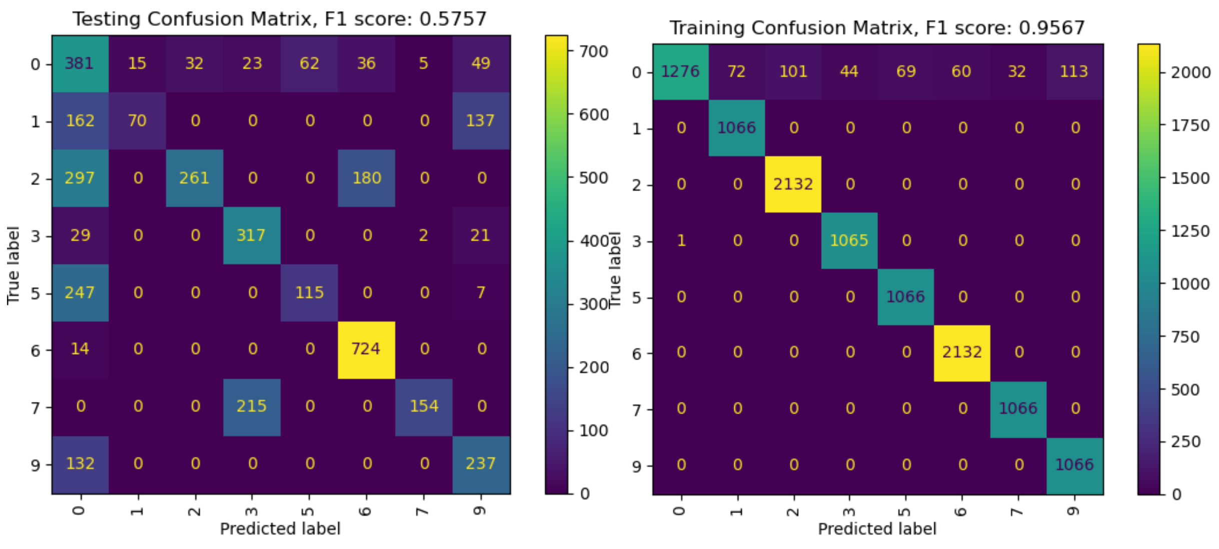


```
              precision    recall  f1-score   support

           0       0.75      0.65      0.70       603
           1       0.95      0.96      0.96       369
           2       0.96      0.98      0.97       738
           3       0.95      1.00      0.97       369
           5       0.83      1.00      0.91       369
           6       0.96      1.00      0.98       738
           7       0.96      0.74      0.84       369
           9       0.93      1.00      0.97       369

    accuracy                           0.91      3924
   macro avg       0.91      0.92      0.91      3924
weighted avg       0.91      0.91      0.91      3924
```

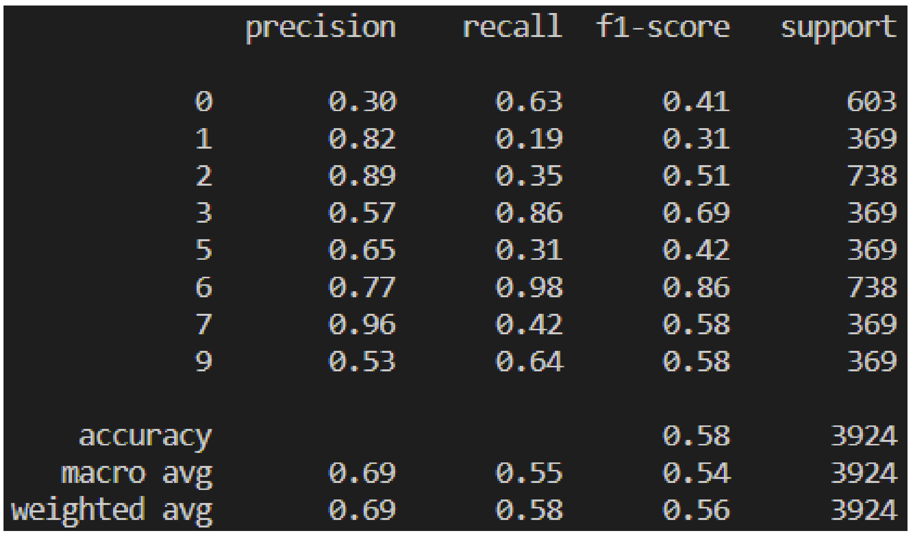
Figure 8



True label \ Predicted label	0	1	2	3	5	6	7	9
0	381	15	32	23	62	36	5	49
1	162	70	0	0	0	0	0	137
2	297	0	261	0	0	180	0	0
3	29	0	0	317	0	0	2	21
5	247	0	0	0	115	0	0	7
6	14	0	0	0	0	724	0	0
7	0	0	0	215	0	0	154	0
9	132	0	0	0	0	0	0	237

True label \ Predicted label	0	1	2	3	5	6	7	9
0	1276	72	101	44	69	60	32	113
1	0	1066	0	0	0	0	0	0
2	0	0	2132	0	0	0	0	0
3	1	0	0	1065	0	0	0	0
5	0	0	0	0	1066	0	0	0
6	0	0	0	0	0	2132	0	0
7	0	0	0	0	0	0	1066	0
9	0	0	0	0	0	0	0	1066

Figure 9



```
              precision    recall  f1-score   support

           0       0.30      0.63      0.41       603
           1       0.82      0.19      0.31       369
           2       0.89      0.35      0.51       738
           3       0.57      0.86      0.69       369
           5       0.65      0.31      0.42       369
           6       0.77      0.98      0.86       738
           7       0.96      0.42      0.58       369
           9       0.53      0.64      0.58       369

    accuracy                           0.58      3924
   macro avg       0.69      0.55      0.54      3924
weighted avg       0.69      0.58      0.56      3924
```

Figure 10

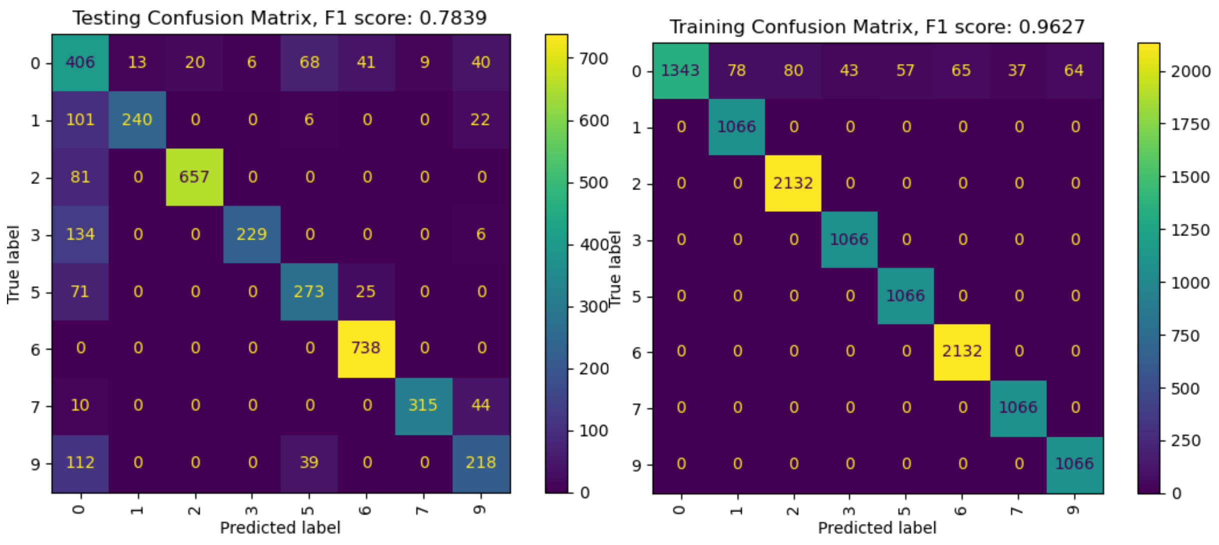
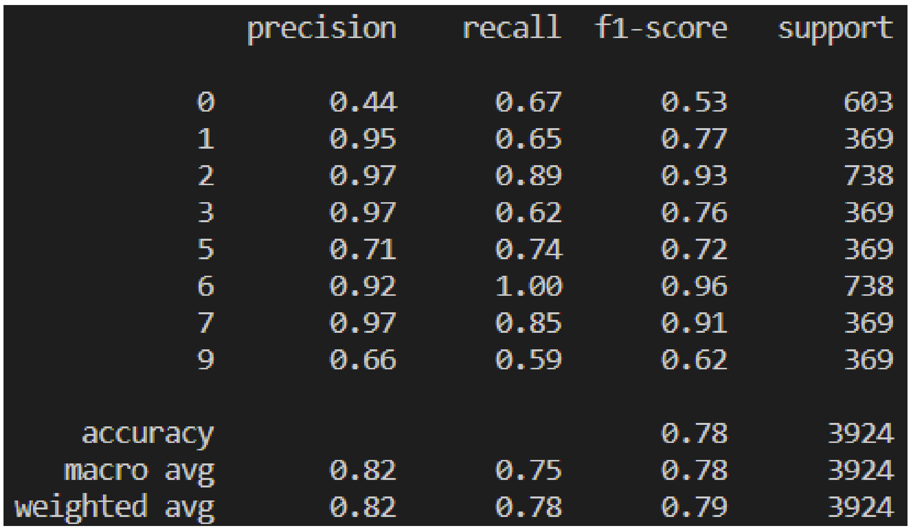


Figure 11



```
              precision    recall  f1-score   support

           0       0.44      0.67      0.53       603
           1       0.95      0.65      0.77       369
           2       0.97      0.89      0.93       738
           3       0.97      0.62      0.76       369
           5       0.71      0.74      0.72       369
           6       0.92      1.00      0.96       738
           7       0.97      0.85      0.91       369
           9       0.66      0.59      0.62       369

    accuracy                           0.78      3924
   macro avg       0.82      0.75      0.78      3924
weighted avg       0.82      0.78      0.79      3924
```

Figure 12

3 Part 3: Question about Central Limit Theorem

3.1 Question 1

Explain what the "central limit theorem" is and provide references to where the concept is covered in your learning materials.

Answer: In the textbook in section 1.6, it says that, the central limit theorem states that the distribution of the sum of a some random variables, from any distribution, approaches a Gaussian distribution as the sample size of the number of random variables being summed approaches infinity. The results of this theorem are used in linear regression (both ML and MAP) where we assume that distribution of samples is chosen from a Gaussian distribution.

3.2 Question 2

For a coin with a probability of heads of 0.6 in each flip, calculate the exact distribution of the total number of heads when flipped 5 times. Provide the equation(s) used to generate the result and a brief explanation of your reasoning.

Answer: This question can be answered by modelling it as a binomial distribution with probability of success $p = 0.5$ and number of trails (coin flips) $n = 5$. The reason for using the binomial distribution is that it is used to model the probability of the number of successes for a given number of trails, which is exactly what this question is asking. The binomial probability density function for this question is:

$$\begin{aligned} Bin(x, n, p) &= {}_nC_x p^x (1-p)^{n-x} \text{ for } 0 \leq \text{x} \leq \text{n} \qquad (34) \\ &= {}_5C_x 0.6^5 (0.4)^{5-x} \qquad (35) \end{aligned}$$

3.3 Question 3

Using the central limit theorem, approximate the distribution of the total number of heads when the coin is flipped 5000 times. Provide the equation(s) used to generate the result and a brief explanation of your reasoning.

Answer: Assuming this question is referring to the coin from question 2, where probability of heads = 0.6.

To get the exact result for this question, we could have used a binomial distribution. However, the question is asking us to use the central limit theorem. An interesting fact about the binomial distribution, is that it becomes similar to a gaussian distribution as the number of trails increases, so the binomial distribution can be thought of as a discrete form of the gaussian distribution. Since the number of trails (coin flips) is reasonably large (5000), we can assume that the resulting binomial distribution very closely resembles a Gaussian distribution. This assumption is also true due to the central limit theorem (even if the trails could not be represented as a binomial distribution, we could have still used the central limit theorem since we are finding the distribution for a sum of random variables).

For a gaussian distribution we need to know the mean and variance. The mean of the binomial distribution is $np = 5000 * 0.6 = 3000$ and the variance is $p(1-p) = 0.6(0.4) = 0.24$. Therefore the gaussian probability densisty function for this problem is:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \tag{36}$$

$$= \frac{1}{.024\sqrt{2\pi}} e^{-(x-3000)^2 / 2*0.24^2} \tag{37}$$

4 Conclusion

I enjoyed working on this project. It was nice being able to implement what we learned in class and work on real data. When coding these models, I also learned how to deal with issues that come up due to floating point errors (I explained these in section 1.4 and 2.4). The coding part of this project was not too difficult, but writing the report (especially the math equations) took longer than I was expecting.