HW1 COMS4771

Xiaoning Wan -- xw2501

Smitha Edakalavan -- se2444

Sravya Pallavi Yandamuri – spy2107

1.

I.

$$\mathcal{L}(\theta|X) = P(X|\theta) = P(x_1, x_2 \dots x_n|\theta) = \prod_{i=1}^{n} \theta e^{-\theta x_i}$$

$$arg\max_{\theta} \mathcal{L}(\theta|X) = arg\max_{\theta} \prod_{i=1}^{n} \theta e^{-\theta x_i} = arg\max_{\theta} \theta^n e^{\sum_{i=1}^{n} -\theta x_i}$$

$$arg\max_{\theta} \mathcal{L}(\theta|X) = arg\max_{\theta} (ln\theta^n e^{\sum_{i=1}^{n} -\theta x_i})$$

$$arg\max_{\theta} \mathcal{L}(\theta|X) = arg\max_{\theta} \left(nln\theta - \sum_{i=1}^{n} \theta x_i \, lne \right) = arg\max_{\theta} \left(nln\theta - \theta \sum_{i=1}^{n} x_i \right)$$

$$\theta_{ml} = \frac{n}{\sum_{i=1}^{n} x_i}$$

II.

$$\mathcal{L}(\theta|X) = P(X|\theta) = P(x_1, x_2 \dots x_n|\theta) = \prod_{i=1}^{n} \frac{1}{\theta} = \frac{1}{\theta^n}$$

$$arg\max_{\theta} \mathcal{L}(\theta|X) = arg\max_{\theta} \frac{1}{\theta^n}$$

As $\theta \geq x$,

$$\theta_{ml} = \max(x_i) \,|_{i \, in \, 1 \, to \, n}$$

III.

$$\mathcal{L}(\theta|X) = P(X|\theta) = P(x_1, x_2 \dots x_n|\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

$$arg\max_{\mu,\sigma^2} \mathcal{L}(\theta|X) = arg\max_{\mu,\sigma^2} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}} = arg\max_{\mu,\sigma^2} log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

$$= arg\max_{\mu,\sigma^2} \sum_{i=1}^{n} \left[-\frac{1}{2} \log(2\pi\sigma^2) - \frac{(x_i-\mu)^2}{2\sigma^2} \right]$$

$$\mu_{ml} = \frac{\sum_{i=1}^{n} x_i}{n},\ {\sigma^2}_{ml} = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$

When we use $\bar{x}$ instead of μ to calculate ${\sigma^2}_{ml}$, we have

$${\sigma^2}_{ml} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} = \frac{1}{n}\sum_{i=1}^{n}((x_i - \mu) - (\bar{x} - \mu))^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}[(x_i - \mu)^2 - 2(x_i - \mu)(\bar{x} - \mu) + (\bar{x} - \mu)^2]$$

$$= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 - 2(\bar{x} - \mu)(\bar{x} - \mu) + (\bar{x} - \mu)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 - (\bar{x} - \mu)^2$$

Thus, unless we have $\bar{x} - \mu$, the ${\sigma^2}_{ml}$ will not be the same as we expected. As introduced, we could use sample variance s^2 instead of variance σ^2 for estimation, where $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$. Or else, we could divide the training data into two parts, and use one part as training data, the other as unbiased test data, so that we can gauge the accuracy of the estimator.

IV.

$$MLE\ \theta = MLE\, g^{-1}\big(g(\theta)\big)$$

$$\theta_{ml} = g^{-1}(g(\theta)_{ml})$$

$$g(\theta_{ml}) = g(\theta)_{ml}$$

Applying this feature, we have

$$\sigma = \sqrt{\sigma^2}$$

And let $g(x) = \sqrt{x}$, then

$$\sigma_{ml} = g(\theta)_{ml} = g(\theta_{ml}) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

2.

I.

$$\mathrm{P}[f_t(x) \neq y] = \mathrm{P}[x > t|y_2]\mathrm{P}[y_2] + \mathrm{P}[x \leq t|y_1]\mathrm{P}[y_1]$$

II.

An optimal threshold would minimize the function

$$\mathrm{f}(x) = \mathrm{P}[x > t|y_2]\mathrm{P}[y_2] + \mathrm{P}[x \leq t|y_1](1 - \mathrm{P}[y_2])$$

that is to minimize

$$\mathrm{f}(x) = \mathrm{P}[Y = y_2 | x > t] + \mathrm{P}[Y = y_1 | x \leq t]$$

suppose that $t = t_0$ is the boundary which satisfies $\mathrm{P}[Y = y_2 | x = t] = \mathrm{P}[Y = y_1 | x = t]$, and $t = t_1$ is the boundary we choose.

When $t_1 > t_0$:

For all $t_1 < x$ and $t_0 > x$, the error function $\mathrm{f}(x)$ has the same value.

For $t_0 < x < t_1$, as $\mathrm{P}[Y = y_2 | x > t_0] < \mathrm{P}[Y = y_1 | x > t_0]$ and if we choose t_0 as boundary we will make decision $Y = y_1$, if we choose t_1 as boundary we will make decision $Y = y_2$, thus we have

$$\mathrm{f}_{t_0}(x) < \mathrm{f}_{t_1}(x) \qquad \text{when} \quad t_0 < x < t_1$$

Thus $\mathrm{f}_{t_0}(x) < \mathrm{f}_{t_1}(x)$

When $t_1 < t_0$:

For all $t_1 > x$ and $t_0 < x$, the error function $\mathrm{f}(x)$ has the same value.

For $t_1 < x < t_0$, as $\mathrm{P}[Y = y_2 | x < t_0] > \mathrm{P}[Y = y_1 | x < t_0]$ and if we choose t_0 as boundary we will make decision $Y = y_2$, if we choose t_1 as boundary we will make decision $Y = y_1$, thus we have

$$\mathrm{f}_{t_0}(x) < \mathrm{f}_{t_1}(x) \qquad \text{when} \quad t_1 < x < t_0$$

Thus $\mathrm{f}_{t_0}(x) < \mathrm{f}_{t_1}(x)$

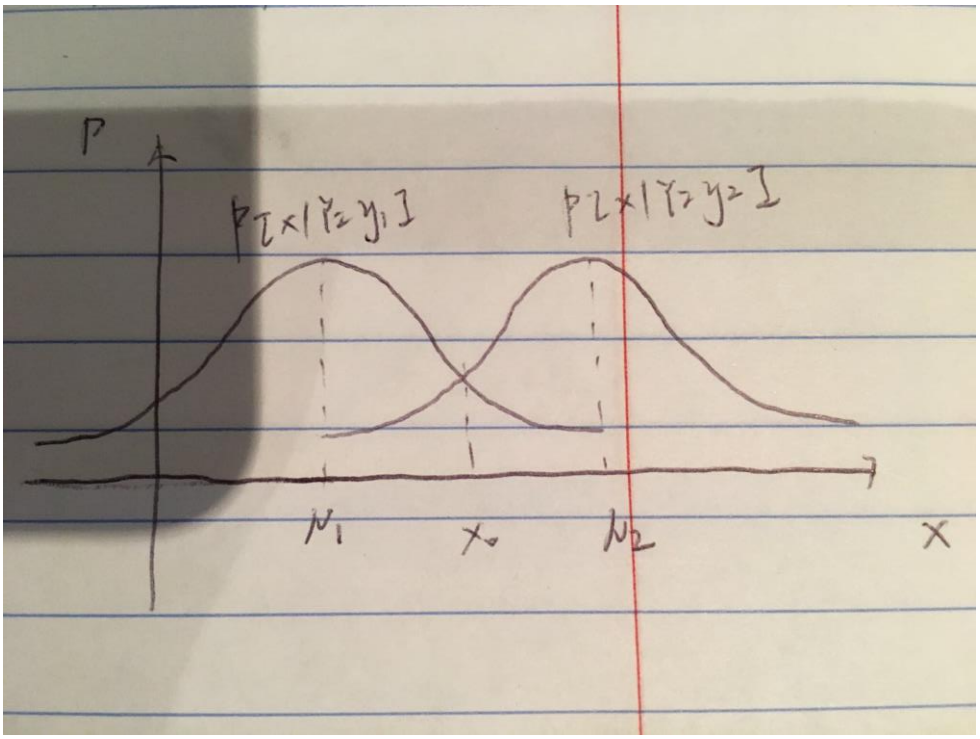
Consequently, the optimal threshold must comply

$$\mathrm{P}[Y = y_2 | x = t] = \mathrm{P}[Y = y_1 | x = t]$$

III.

When the Gaussian distribution of these two classes have the same σ^2 , there **exists** a threshold value t that could have rule f_t and Bayes have the same error rate.

i.e. when $\sigma^2{}_{y1} = \sigma^2{}_{y2}$ probability function of these two classes will look like this, x_0 is the point where $P[X|Y = y1] = P[X|Y = y2]$



In this case, if we choose threshold $t = x_0$, and make decision as when $x > t$, $Y = y2$ else $Y = y1$. Thus we could calculate f_t

$$f_t(x) = \int_{-\infty}^{t} P[X|Y = y1]\, P[Y = y1] + \int_{t}^{+\infty} P[X|Y = y2]\, P[Y = y2]$$

$$= \frac{1}{2}\left(\int_{-\infty}^{x_0} P[X|Y = y1] + \int_{x_0}^{+\infty} P[X|Y = y2]\right)$$

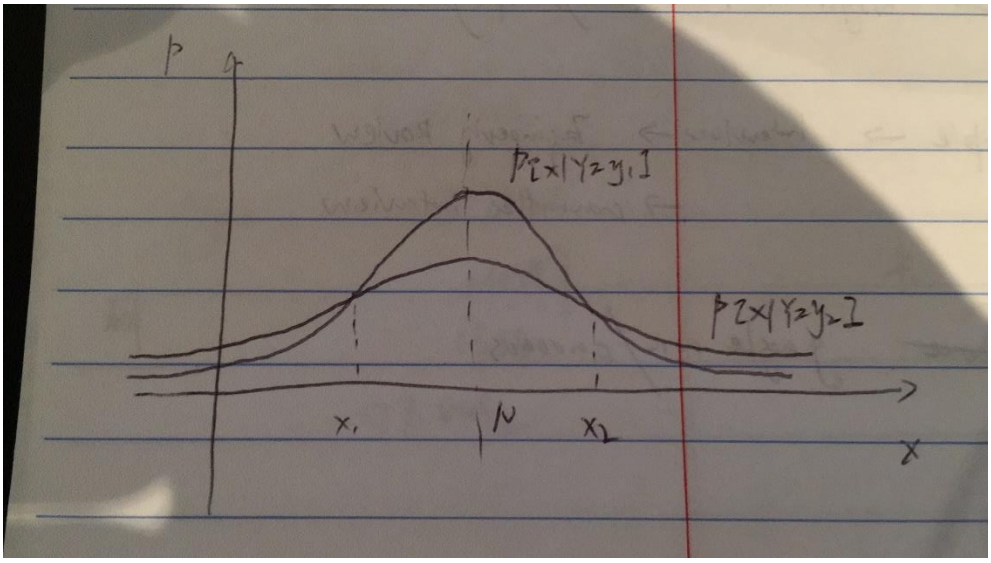
And the error rate of f_{Bayes}

$$f_{Bayes} = \int_{-\infty}^{x0} P[X|Y = y1]\, P[Y = y1] + \int_{x0}^{+\infty} P[X|Y = y2]\, P[Y = y2] = f_t$$

When the Gaussian distribution of these two classes have different σ^2 , there **doesn't exist** a threshold value t that could have rule f_t and Bayes have the same error rate.

i.e. we may assume $\mu_{y1} = \mu_{y2} = \mu$ and ${\sigma^2}_{y1} < {\sigma^2}_{y2}$, as a matter of fact, the value of μ doesn't really affect the result.

the probability function of these two classes will look somewhat like this, x_1 and x_2 are the point where $P[X|Y = y1] = P[X|Y = y2]$



We could choose t as the threshold, we may decide if $x > t, Y = y1$ else , $Y = y2$, thus the error rate of f_t

$$f_t(x) = \int_{-\infty}^{t} P[X|Y = y1]\, P[Y = y1] + \int_{t}^{+\infty} P[X|Y = y2]\, P[Y = y2]$$

$$= \frac{1}{2}\left(\int_{-\infty}^{t} P[X|Y = y1] + \int_{t}^{+\infty} P[X|Y = y2]\right)$$

While error rate of f_{Bayes}

$$f_{Bayes} = \int_{-\infty}^{x1} P[X|Y = y1]\, P[Y = y1] + \int_{x1}^{x2} P[X|Y = y2]\, P[Y = y2] + \int_{x2}^{+\infty} P[X|Y = y1]\, P[Y = y1]$$

$$= \frac{1}{2}\left(\int_{-\infty}^{x1} P[X|Y = y1] + \int_{x1}^{x2} P[X|Y = y2] + \int_{x2}^{+\infty} P[X|Y = y1]\right)$$

Once we have a look at the image above, it is quite obvious that f_t is always bigger than f_{Bayes} no matter what threshold we choose because when x<x1 or x>x2, $P[X|Y = y2]$ is always bigger than $P[X|Y = y1]$.

3.

I.

We may assume that at a specific location state x, the expected reward is K, thus the expected reward (note that x us continuous), $E_{x,a}[f] = \int KP[x]dx$.

Now we need to calculate K. As given in the question, reward K is based on the state x and action a, and a is randomly chosen with probability P[a|x] (note that a is discreet),

$$K = \sum_{a} R(x, a)P[a|x]$$

Thus, we have

$$E_{x,a}[f] = \int KP[x]dx = \int \{\sum_{a} R(x, a)P[a|x]\}\, P[x]dx$$

II.

For a given state x, the expected reward is given as

$$E_{x,a}[f] = \int \{\sum_a R(x,a)P[a|x]\}\, P[x]dx$$

When we choose $P\left[\left(a = argmax\big(R(x,a)\big)\right)\middle|x\right] = 1$ as our decision, the expected reward

$$E_{a=argmax(R(x,a))}[f] = \int \max(R(x,a))P[x]dx$$

We may assume another random decision, the expected reward would be

$$E_{random}[f] = \int \{\sum_a R(x,a)P[a|x]\}\, P[x]dx$$

Subscribe $E_{random}[f]$ from $E_{a=argmax(R(x,a))}[f]$, we have

$$E_{a=argmax(R(x,a))}[f] - E_{random}[f] = \int [\max\big(R(x,a)\big) - \{\sum_a R(x,a)P[a|x]\}]P[x]dx$$

Where $[\max\big(R(x,a)\big) - \{\sum_a R(x,a)P[a|x]\}] \geq 0$, thus

$$E_{a=argmax(R(x,a))}[f] \geq E_{random}[f]$$

Consequently, for a given state x, $P\left[\left(a = argmax\big(R(x,a)\big)\right)\middle|x\right] = 1$ is an optimal decision.

III.

Yes, when the agent need to make a series decisions and to maximize the total reward, randomizing a suboptimal rule will be benefit.

To illustrate this idea, I will make an example

+5	-5	-10
+5	-10	+20
-10	+20	+20

In this case, the states are not continuous but discreet, but it still shows the same idea. Each block in this table means a state, the number in it is the reward for moving into this block, at each block, the possible actions are moving vertically or horizontally, and the chances for all possible decisions are equal.

If we assume an agent with start state at the block marked red, and we want it to make 10 decisions so that we can obtain max reward.

If we use the policy mentioned above, it will move up first, and then it will go down and then repeat. Apparently this will not be the optimal policy. In this case, we want it to move to the down-right corner so that it could obtain maximum value, thus randomizing a suboptimal rule would be beneficial.

4.

I.

Based on the assumption, we may suppose that $b = a + t(t \neq 0)$, thus we can rewrite it as

$$|f'(a+t) - f'(a)| \leq \mathrm{L}|t|$$

$$\left|\frac{f'(a+t) - f'(a)}{t}\right| \leq \mathrm{L}$$

Let $t \to 0$, then

$$\left|\lim_{t\to 0}\frac{f'(a+t) - f'(a)}{t}\right| \leq \mathrm{L}$$

$$|f''(a)| \leq \mathrm{L}$$

Which means that f has bounded second derivative L.

Now, analyze $f(x) - f(\bar{x})$

$$f(x) - f(\bar{x}) = f(x) - \left[f(x) + f'(x)(x - \eta f'(x) - x) + \frac{1}{2}f''(z)(x - \eta f'(x) - x)^2\right]$$

$$= \eta f'^2(x) - \frac{1}{2}f''(z)(\eta f'(x))^2$$

$$= f'^2(x)\left[\eta - \frac{1}{2}f''(z)\eta^2\right]$$

$$= f'^2(x)\left[1 - \frac{1}{2}f''(z)\eta\right]\eta$$

As $f''(z)$ is bounded $[-L, L]$,

If $f''(z) \leq 0$, then for any $\eta > 0$, $f(x) - f(\bar{x}) \geq 0$ and only when $f'(x)$ we have $f(x) - f(\bar{x}) = 0$

Else if $f''(z) > 0$, then for $0 < \eta < \frac{2}{f''(z)}$, $f(x) - f(\bar{x}) \geq 0$ and only when $f'(x)$ we have $f(x) - f(\bar{x}) = 0$

II.

As computed in I, the no matter what value x is, so long as $0 < \eta < \frac{2}{f''(z)}$, the relationship will comply $f(x) \geq f(\bar{x})$. And as $f''(z)$ is bounded $[-L, L]$, we can simply set $\eta < \frac{2}{L}$.

In this case, we may set $\eta = \frac{1}{f''(x)}$. And as the algorithm must halt at a specific point, we may set the threshold to be 0.00001.

Designed algorithm:

Note that the algorithm is **not** in any kind of programming language.

start = x0;

threshold = 1e-5;

step = 1/derivate (derivate(f, start), start)* derivate(f, start);

while(diff>threshold) {

start = start – step;

step = 1/derivate (derivate(f, start), start)* derivate(f, start);

diff = step*derivate(f, start);

}

return start;

III.

This the designed algorithm in matlab format, I will also include the matlab script in the homework.

```
% as function f(x) = (x-3)^2 + exp(x)
% the corresponding devrivation is 2(x-3) + exp(x)
% the second devrivation is 2 + exp(x)
start = 500;                                                % start point
threshold = 1e-5;                           % threshold for halt decision
diff = (1/(exp(start) + 2))*(2*(start - 3) + exp(start));    % step size
cnt = 0;                                                    % loop count
while abs(diff)>threshold
    start = start - diff;
    diff = (1/(exp(start) + 2))*(2*(start - 3) + exp(start));
    cnt = cnt + 1;
end
end_piont = start                                            % end point
cnt                                                         % loop times
```

The minimum value is found at theta = 1.2518. And the minimum value is 6.5528

5.

Here we will only give a brief introduction of the scripts. For more details, please refer to the notes in scripts.

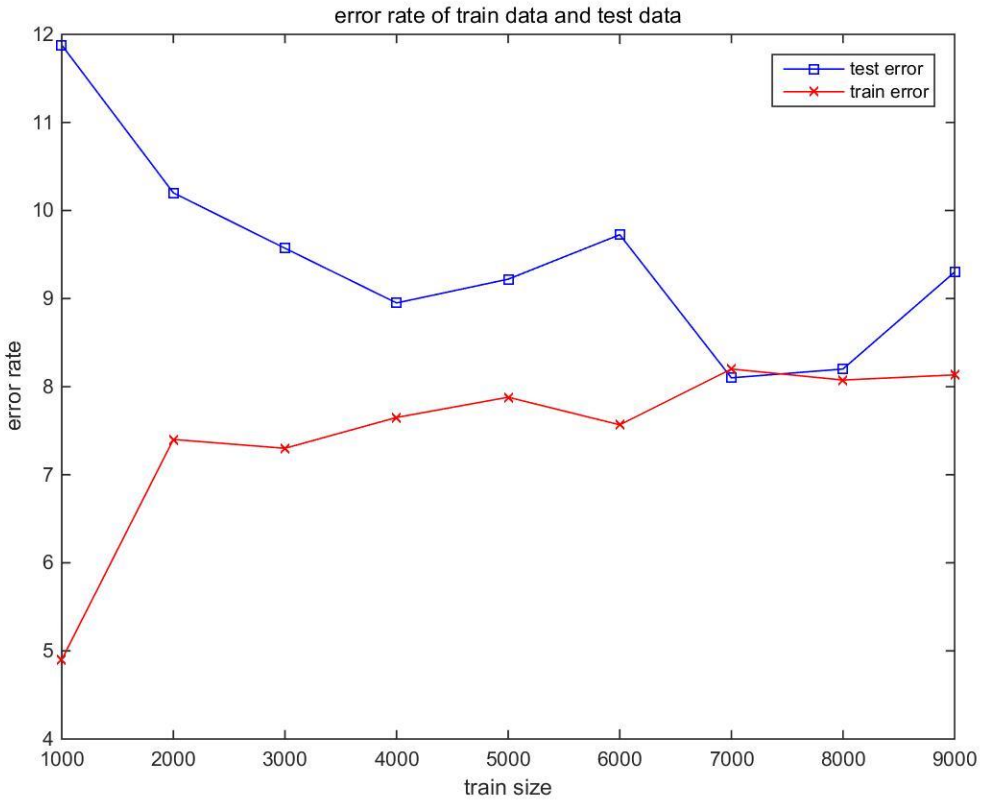
I.

Before building the classifier, we need first to take a look at the variance of the multivariate Gaussian distribution of the images. As there is some blank area in the image, the variance of these pixels would be zero, hence there will be some rows and columns in matrix $\sum ML$ where all the members are zero. We could not use this singular matrix directly.

To solve this problem, we wrote a script to do some preprocessing to the images. We first find the boundary of the digit in the image, and then we extract the part of image with digit. Then, we scale the extracted part into a 4X4 size image, that is to compress the dimension from 784 down to 16, which will much fasten the computing speed. The scaling process is done by compute the mean value of a block in origin image and use it as the value of the corresponding pixel in the compressed image.

We also built another classifier according to naïve Bayesian classifier. However, the performance of this classifier is not as good as former one, thus we will not discuss the build of naïve Bayesian classifier. Also, the script of this classifier is noted, it will not affect other parts of the script.

Here is the running result of multivariate Gaussian mode classifier:

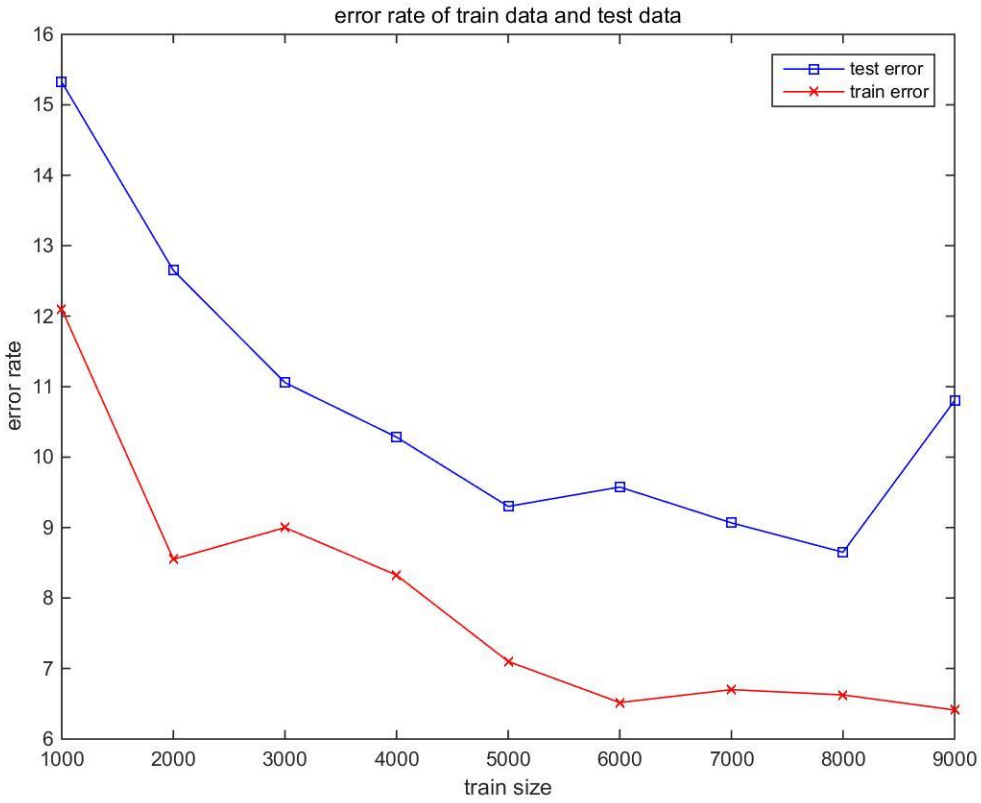


II.

k-NN classifier is rather simple. We just need to calculate the Euclidean metric between test data and all the training data, then do a sort, and find the label that appears most times in the first k sorted array, and we have a prediction result.

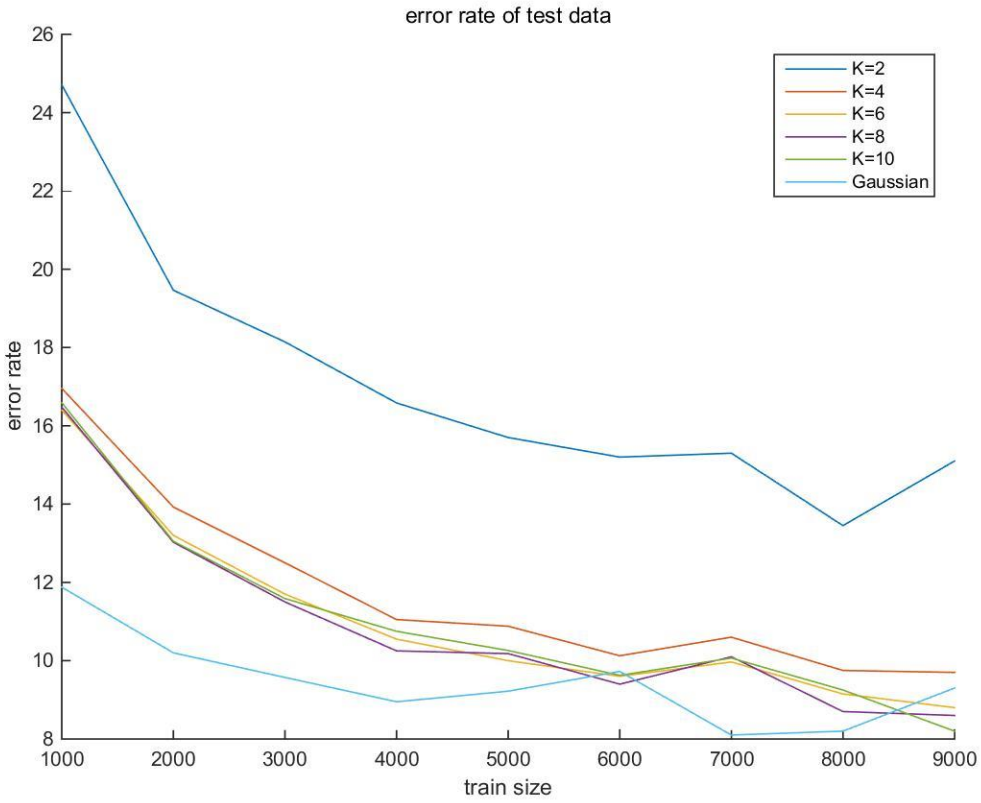
The idea is pretty simple but we can't do it directly, as the dimension of the image is too large, calculating the distance will cost a long time. Thus, we also did a same preprocess as that in multivariate Gaussian model classifier. By reducing the dimensions, the running speed will increase dramatically, though it is still much slower than multivariate Gaussian model.

Here is the result of k-NN classifier(k=5)



III.

Several factors will affect the performance of k-NN classifier. For this question, we will discuss two factors size of training data and the value of k.



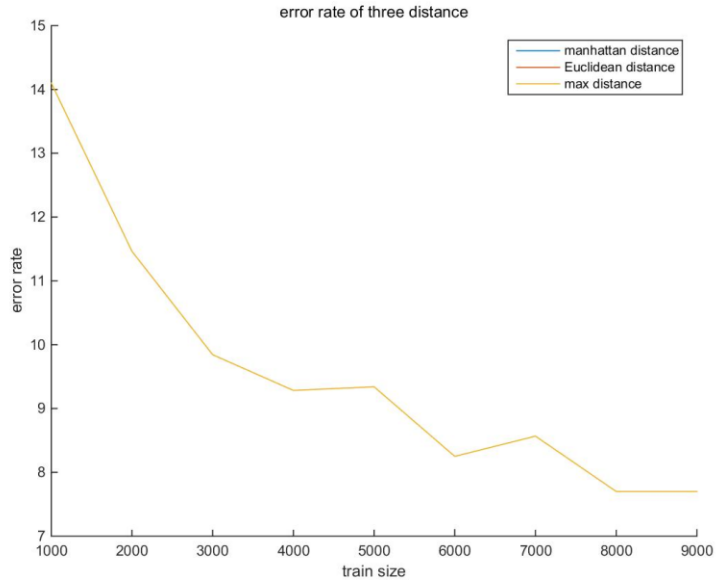
As shown above, the trend of error rate and train size is generally the same, error rate falls as train size grows.

And we could see that as K increases, the error rate of k-NN gradually decreases. However, it is still higher than multivariate Gaussian model classification.

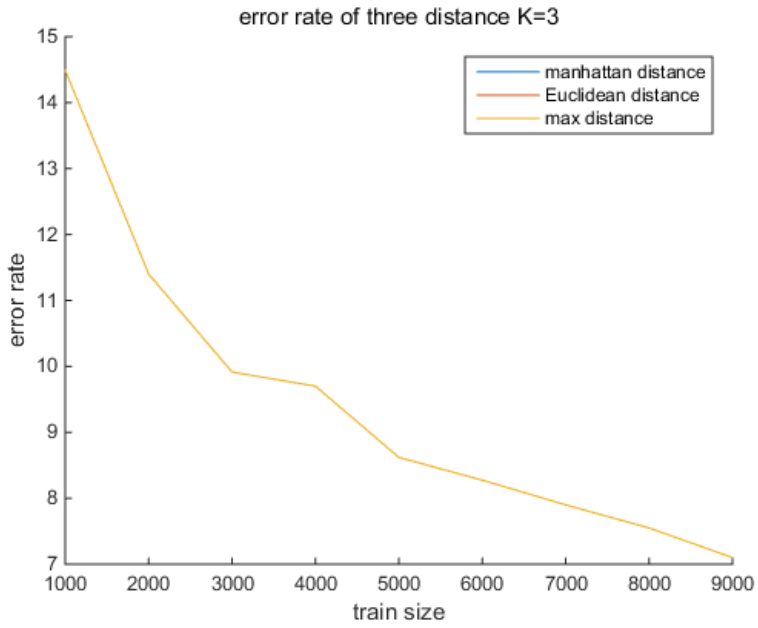
As for the running speed, multivariate Gaussian model classifier is also better, it has a much faster speed. (we did not draw an elapsed time graph, but you could run the script and see the elapsed time in the command line)

There is indeed one thing k-NN is better, it is less complex. In multivariate Gaussian model classifier, we need to compute the determinant and the inverse matrix of $\sum ML$. In this case, $\sum ML$ is a 16 by 16 matrix, thus it is relatively easy to compute these two value. However, we need to compress the image to achieve it, if we could not compress the image because we want all the information of the image to be preserved, we may find it difficult to compute these two value in a large matrix. Another problem is that the matrix $\sum ML$ is sometimes singular or has a determinant too big we can store it in software, we could no longer use this algorithm. While k-NN could also work in this situation.

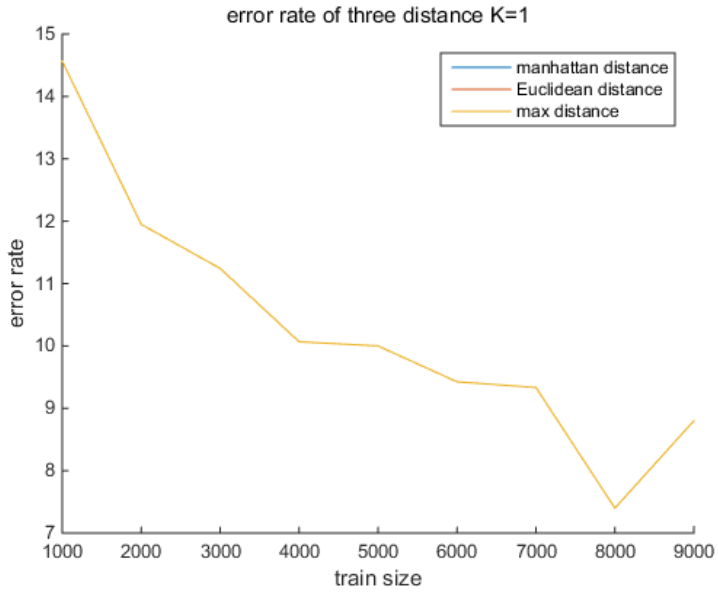
IV.



When k = 5, Error rate of three kinds of distance in different train size are identically the same.



The result is the same for k=3.



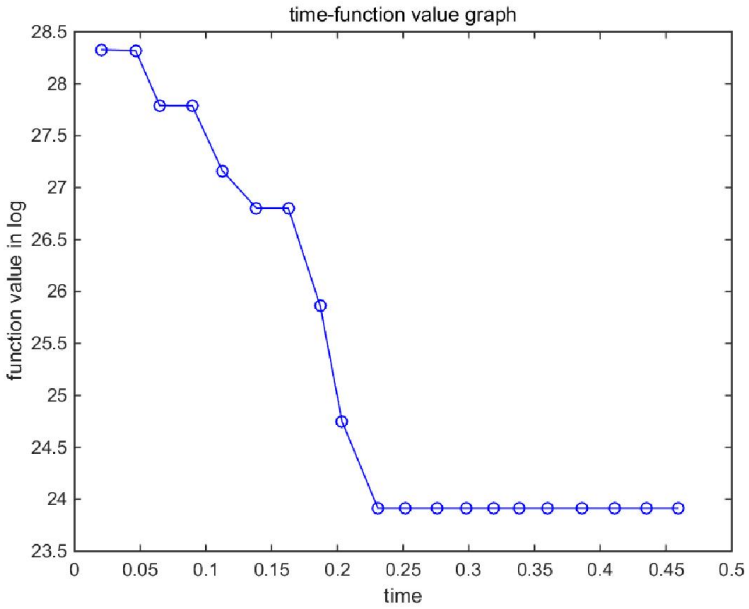
Also, when k = 1, this result still holds. We may conclude that in this problem, these three kinds of distance have the same performance regarding of accuracy.

6.

I.

$$\nabla_\theta f = \sum_{x,y} \sum_{k} \sum_{d} -y_k^i (x_d^i - \theta_k)$$

II.

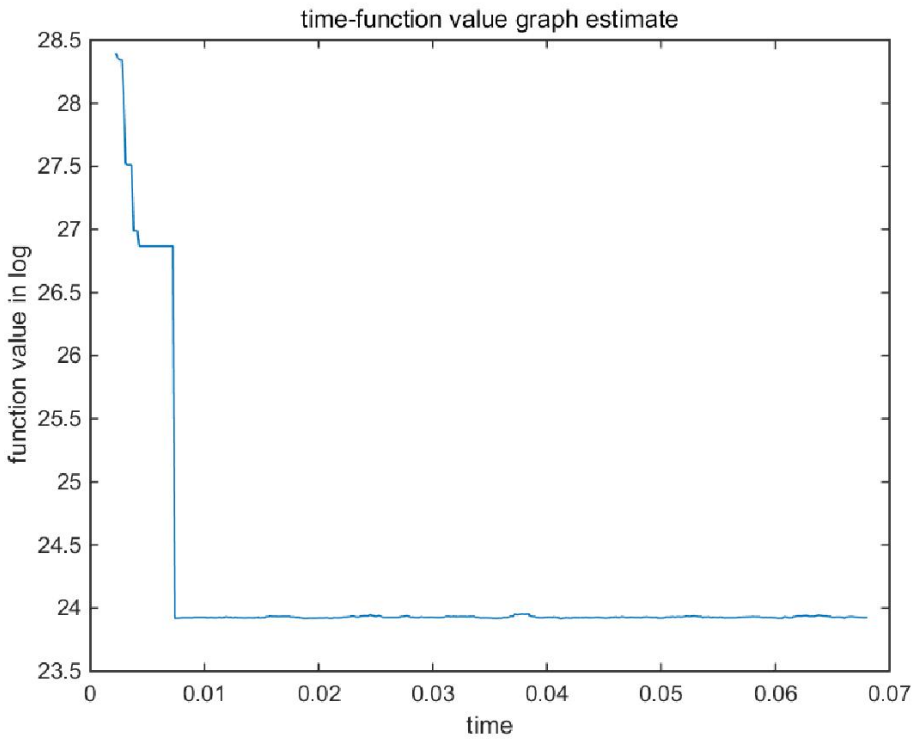


This is the graph of time-function value, the unit of time is second(s) and the function values are in logarithm(log). The reason for using logarithm to represent function value is that the origin function value is very big, using logarithm would have a better view of the trend of function value.

III.

$$\nabla_\theta f_i = \sum_k \sum_d -y_k^i (x_d^i - \theta_k)$$

IV.

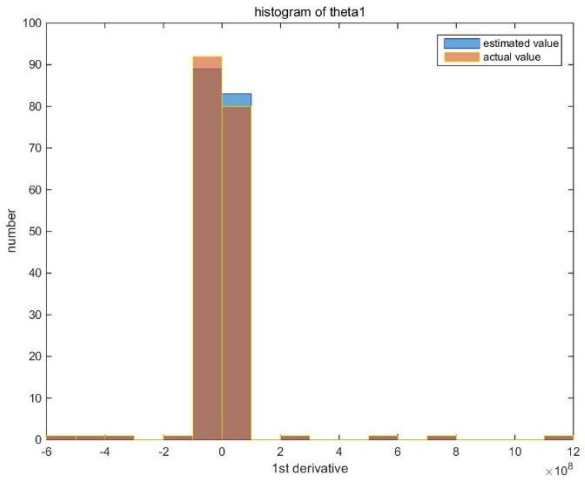


Time-function value of graph of this method. The unit of time is also second(s) and the function value is also in logarithm for the same reason as part II.

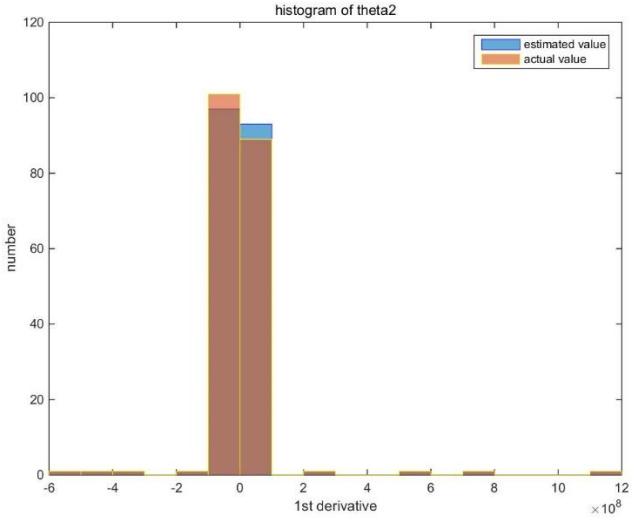
V.

Here is the histogram of estimated value and actual value:

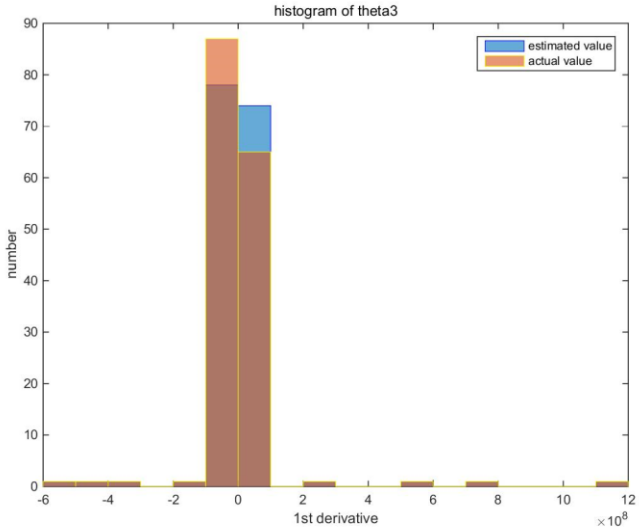
θ_1



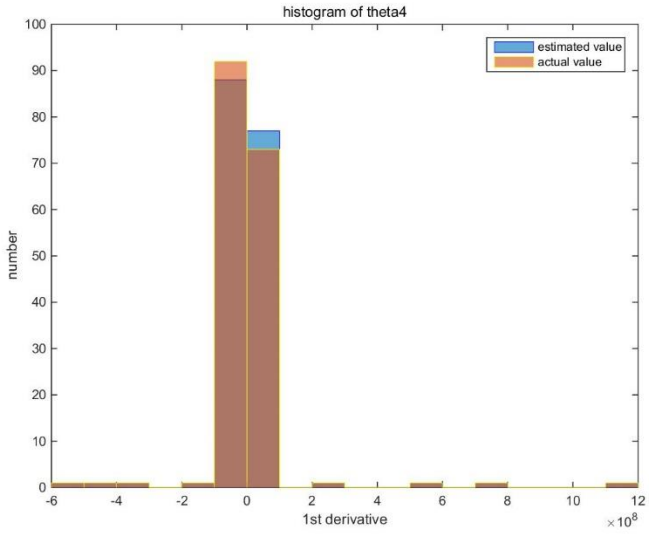
θ_2



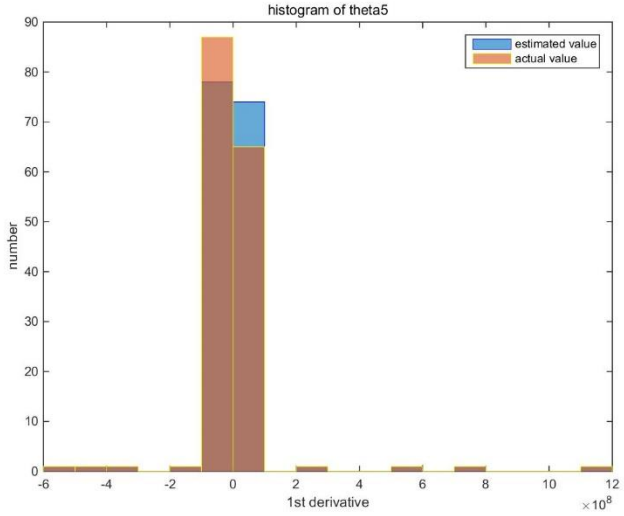
θ_3



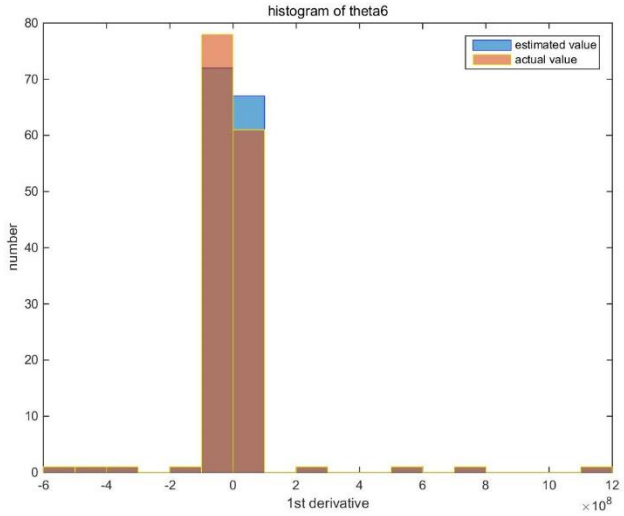
θ_4



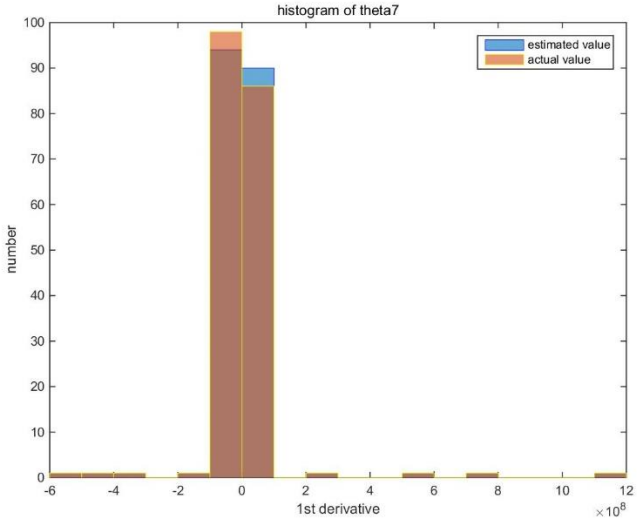
θ_5



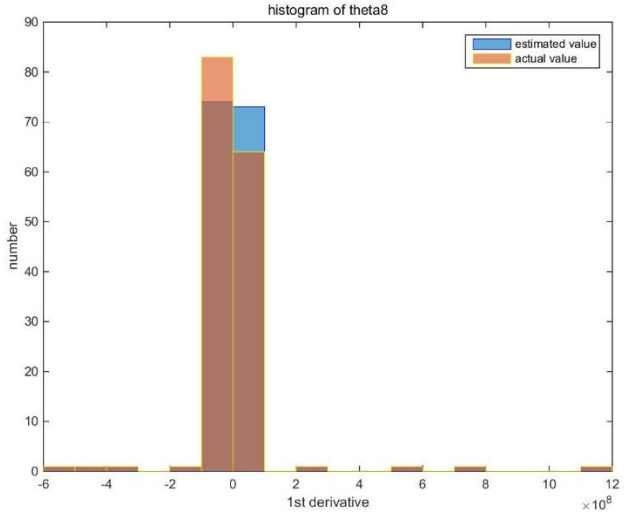
θ_6



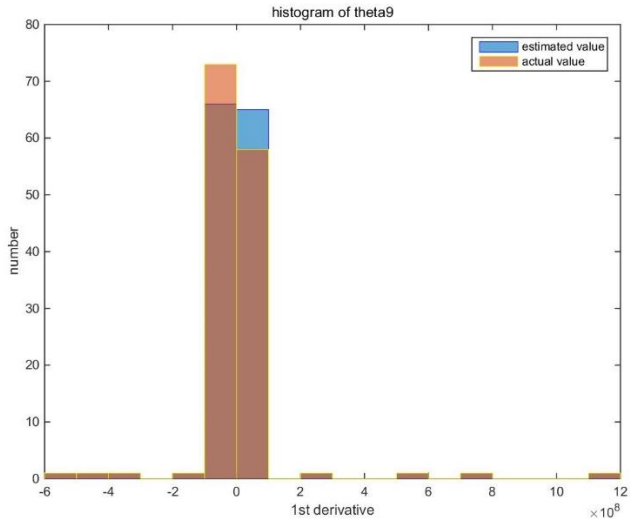
θ_7



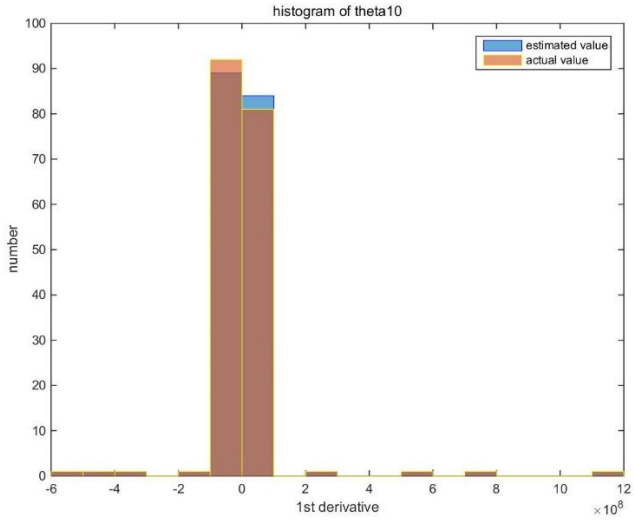
θ_8



θ_9



θ_{10}

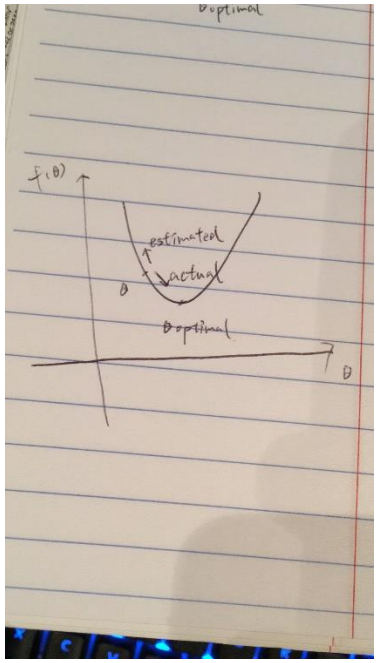


As illustrated above, the histogram of ten dimensions of estimated gradient and actual gradient have very similar trend, with a large number of gradients concentrated to zero and a relatively small number of sparse gradients.

The distributions of sparse gradients are the same between estimated value and actual value. While the concentrated part has an interesting difference, the concentrated part of estimated value is more evenly distributed and the actual value seems to vary a little more.

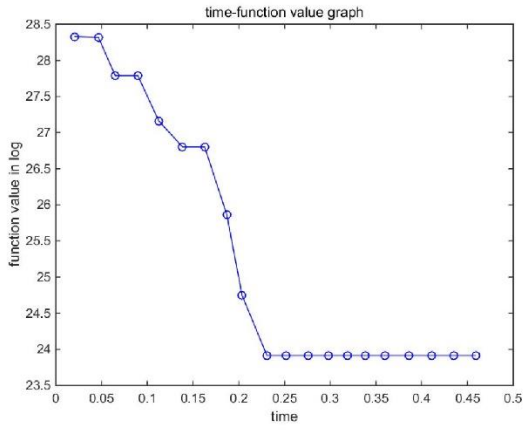
When the theta is 'far away' from the optimal value, $\nabla_\theta f$ depends more on θ than x_d^i. Thus, although the arbitrary sample may deviate from the mean value, as it has little influence in $\nabla_\theta f$, we can still use it as an approximation of full gradient.

However, when the theta is 'near' to the optimal value, x_d^i starts to be more important in $\nabla_\theta f$. And this arbitrary value may not be able to represent full gradient.

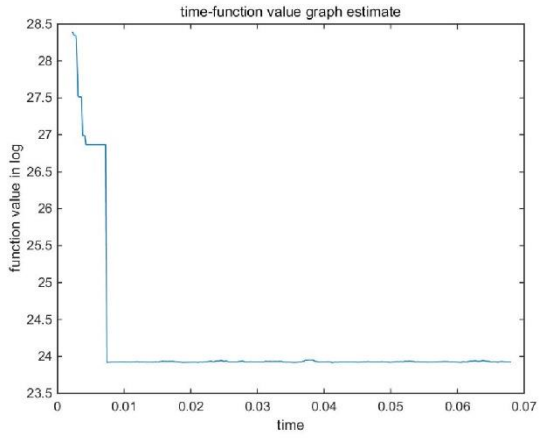


For example, this picture above is one dimension of θ that we are trying to optimize. Now it is at the θ point as shown. According to the actual gradient, we should increase θ to achieve the optimal, but as our sample is randomly chosen, there is a chance that our estimated gradient guide us to an opposite direction. In this progress, the actual gradient is negative and the estimated gradient is positive. This explains why the distribution of gradients close to zero varies between actual value and estimated value.

VI.



result of part II



result of part IV (unit of time is also second and value is also in log)

It is obvious that the method of part IV takes much less time than method of part II due to the reduction of computation times.

Also we may notice that as the theta getting close to the optimal value, it takes more proportion of time for method of IV to achieve the optimal value. And in fact, the 'optimal theta' obtained in this method is not exactly the optimal value, there is a slight difference between the actual optimal theta and our achieved theta. This is also because that our gradient is estimated by an arbitrary sample from the whole dataset. When the theta is close to the optimal value, x_d^i becomes more important in $\nabla_\theta f$. Thus it will take a larger proportion of time to have a step size smaller than the threshold to halt the iteration.

7.

I.

Before building the decision tree, we also first processed the data X with preProcess.m introduced in Q5.

The main idea is to find a feature and corresponding threshold which minimizes the uncertainty, in this program we choose Entropy, of the data. Thus, we need a nest loop to realize this process, outer loop is all the features (9 pixels), and the inner loop is to find the threshold of each feature. With this nest loop, we will find the ideal feature and threshold.

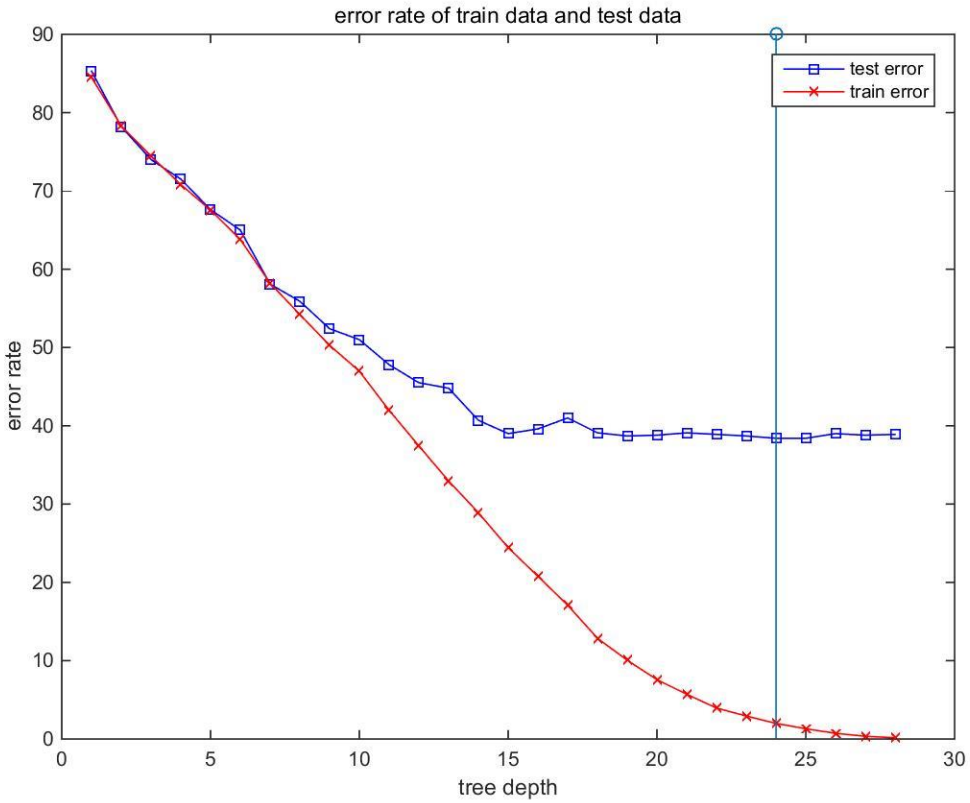
And we will continue this procedure until the 'tree' achieves the 'tree_depth'. As the tree grows, we also save the selected features and thresholds in an array, which will later be used for testing. There are some detail parts we designed to save running time such as fillResult.m script. This script is designed to fill all the subtree when we reach a node where there is only one label left. In this case, we do not need to find features any more, then we could use this function to fill all the subtrees and the result.

As for the test part, what we need to do is to let the teat data 'go along the tree'. We compare the feature of test data to the threshold to decide which subtree to go. After that the test data will compare with feature of next depth. When the depth of test data reaches the depth of tree, we have a prediction.

For more details, please refer to notes of the scripts.

II.

The light blue line indicates the train size where the test error is the minimum value.



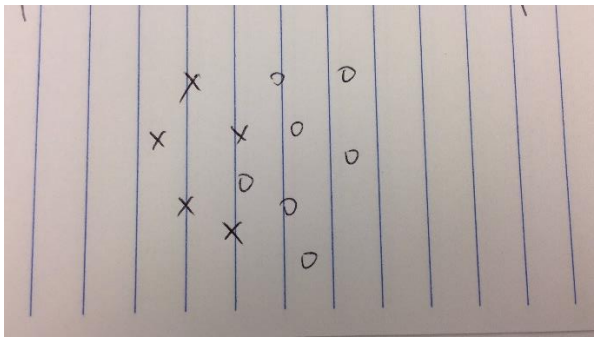
III .

Our script was designed to randomly choose training data from all the data. And during our testing part, we did not find any change of accuracy due to the split of data.

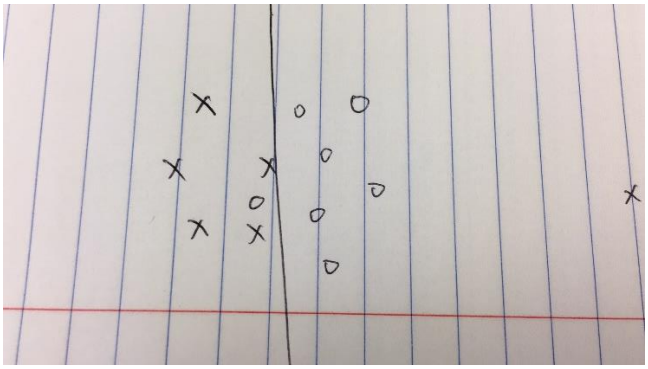
IV.

As tree depth grows, the training error will keep decreasing while the testing error will decrease first and then increase at some point.

It is because of the phenomenon of overfitting. For example, if we would like to use decision tree to classify two class marked as ‘X’ and ‘O’



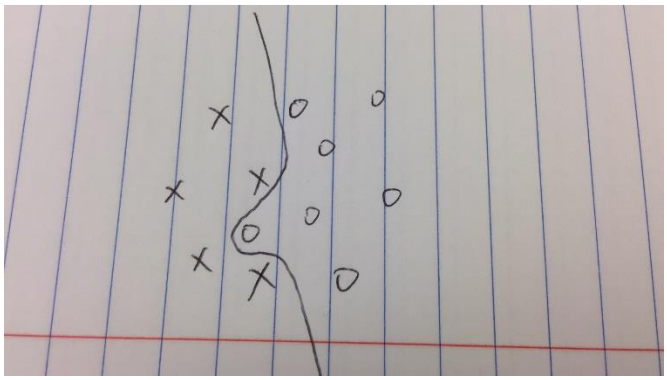
The ideal boundary would be like this



The data are classified according to the distribution of the majority, and some data deviated from the majority due to noise or some other reason do not affect the classification boundary.

When the depth of decision tree is not too large, the feature of classification will be the feature which minimizes the uncertainty, thus these points could not affect the boundary.

This is the situation of decision tree when depth is not too large. However, when the depth is too large, the boundary would be like



At this time, the feature which minimizes the uncertainty would be the feature excludes the single 'O' from around 'X'. And the boundary becomes this shape.

The problem is that we are letting the noise points affect our boundary, in this case, although the boundary could classify the training data well, it does not fit the real situation any more. This is the why there is overfitting phenomenon.

V.

If we need the highest accuracy of prediction, the ideal depth K would be the point where error rate of test data is minimum, that is $K = \min_K Error\_rate_{test}$. In the situation of Q2, is K = 24.

If we also have other consideration such as time and complexity of algorithm, we should choose a smaller K, that is $K < \min_K Error\_rate_{test}$.