1. (a) when $l(f(x), y) = (f(x) - y)^2$

$$E[l(f(x), y)] = \int_x \int_y (f(x) - y)^2 P(x,y)\, dy\, dx$$

$$= \int_x \int_y (f(x) - E(y|x) + E(y|x) - y)^2 \cdot P(x,y)\, dy dx$$

$$= \int_x \int_y [(f(x) - E(y|x))^2 + (y - E(y|x))^2 - 2(f(x) - E(y|x))(y - E(y|x))] P(x,y)\, dy\, dx$$

Let's look at the last term:

$$\int_x \int_y (f(x) - E(y|x))(y - E(y|x)) P(x,y)\, dx dy$$

$$= E(f(x)) \cdot E(y) - E(f(x)) E(y) - E(y) \cdot E(y) + E^2(y)$$

$$= 0$$

Since we can not modify y or $E(y|x)$, In order to minimize $E[l(f(x), y)]$

$f(x) = E(y|x)$. Conditional Expectation of y

(b) when $l(f(x), y) = |f(x) - y|$

$$E[l(f(x), y)] = \int_x \int_y |f(x) - y| P(x,y)\, dx dy$$

$$= \int_x \underbrace{\left(\int_y |f(x) - y| P(y|x) \cdot dy\right)} \cdot P(x) \cdot dx$$

Because x can be chosen independent of $f(x)$, we can ignore integral over x and instead focus on how to minimize curly brace

If we take dirivative and set to 0:

$$\frac{\partial}{\partial f(x)} \int_y |f(x) - y| P(y|x)\, dy$$

$$= \frac{\partial}{\partial f(x)} \left[\int_{-\infty}^{f(x)} (f(x) - y) P(y|x) \cdot dy + \int_{f(x)}^{+\infty} (y - f(x)) P(y|x) \cdot dy\right] = 0$$

$$\Rightarrow \int_{-\infty}^{f(x)} P(y|x)\, dy = \int_{+\infty}^{f(x)} P(y|x)\, dy$$

Optimal $f(x)$ is the conditional median

2 Correct

Let us consider a classifier $f \neq f^*$

$P(f \neq y|x)$

$= 1 - P(f=1 \,\&\, y=1|x) - P(f=-1 \,\&\, y=-1|x)$

$= 1 - P(f=1|y=1,x) \cdot P(y=1|x) - P(f=-1|y=-1,x) \cdot P(y=-1|x)$

Because for each (x, y) $f(x)$'s value can be chosen independently of y

$= 1 - P(f=1|x) \cdot P(y=1|x) - P(f=-1|x) \cdot P(y=-1|x)$

Because (x, y) is drawn from fixed distribution D, Let $\alpha(x)$ be

$\alpha(x) = P(y=1|x) = 1 - P(y=-1|x)$

$P(f \neq y|x) = 1 - \mathbb{1}(f=1) \cdot \alpha(x) - \mathbb{1}(f=-1) \cdot (1-\alpha(x))$

$P(f \neq y|x) - P(f^* \neq y|x)$

$= \alpha(x) \cdot [\mathbb{1}(f^*=1) - \mathbb{1}(f=1)] + (1-\alpha(x))[\mathbb{1}(f^*=-1) - \mathbb{1}(f=-1)]$

Note $\mathbb{1}(f^*=-1) = 1 - \mathbb{1}(f^*=1)$, $\mathbb{1}(f=-1) = 1 - \mathbb{1}(f=+1)$

$= \alpha(x) \cdot [\mathbb{1}(f^*=1) - \mathbb{1}(f=1)] - (1-\alpha(x))[\mathbb{1}(f^*=1) - \mathbb{1}(f=1)]$

$= (2\alpha(x) - 1)[\mathbb{1}(f^*=1) - \mathbb{1}(f=1)]$

Note $f^*(x) = \begin{cases} 1 & \text{if } \alpha(x) \geq \frac{1}{2} \\ -1 & \text{else} \end{cases}$

For x such that $\alpha(x) \geq \frac{1}{2}$,

$$\underbrace{P(f \neq y|x) - P(f^* \neq y|x)}_{\geq 0} = \underbrace{(2\alpha(x) - 1)}_{\geq 0}\underbrace{[\underbrace{\mathbb{1}(f^*=1)}_{=1} - \underbrace{\mathbb{1}(f=1)}_{=0 \text{ or } 1}]}_{\geq 0}$$

For x such that $\alpha(x) < \frac{1}{2}$

$$\underbrace{P(f \neq y|x) - P(f^* \neq y|x)}_{\geq 0} = \underbrace{(2\alpha(x) - 1)}_{<0}\underbrace{[\underbrace{\mathbb{1}(f^*=1)}_{0} - \underbrace{\mathbb{1}(f=1)}_{=0 \text{ or } 1}]}_{\leq 0}$$

Thus $P(f \neq y|x) - P(f^* \neq y|x) \geq 0$ always hold

Conclusion $L(f^*) \leq L(f)$ proved.

3:
(i)
First calculate between class S_b,within class S_w

$$S_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$
$$S_w = \sum_{\mathbf{x}_n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{\mathbf{x}_n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

Then
calculate the largest absolute eigenvalue as well as corresponding eigenvectors for inv(S_w)*S_b
Use this eigenvector to project X and discriminant value by left multiply w^T, and discriminant value is set as the average of two class mean vectors. And then compare the projected X value and project discriminant value.
train error rate: 0.310283625731
test error rate: 0.310785714286
test error standard deviation: 0.220061320304
We can not project Boston50 dataset to 2D subspace, because the S_B is composed of sum of K matrices, each of which is an outer product of two vectors and thus S_B has rank 1. In addition, only (K-1) matrices are independent due to constraint that overall mean =weighted average of each class. So S_B has at most (K-1) rank and at most (K-1) nonzero eigenvalues. Because we need projection matrix to project onto subspace, we can not find more than (K-1) eigenvectors to compose the projection matrix and can not find more than (K-1) dimensional subspace.
(ii)
Do the same calculations of S_b and S_w, except the S_w consists of sum of ten classes.
Because we want to project to 2D dimensional subspace, we choose two eigenvectors with largest eigenvalues.
Stack two eigenvectors horizontally as projection matrix to project whole dataset X
In 2D subspace, calculate the covariance matrix and mean using bi_vairate joint density function

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu_k)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu_k)\right\}$$

P(C_k|x) is proportional to P(x|C_k)*P(C_k)
Prior P(C_k)=N_k/N
After we calculate covariance and mean, we can use them to predict P(C_k|x) and choose the k corresponding to the largest posterior probability.
train error rate: 0.817658457499
test error rate: 0.830756292425
test error standard deviation: 0.0421947670347
4.

Summary of algorithm

(i)Logistic regression:

FIrst, I will check the number of class K is larger than 2 or not.

I used the iterative update approach called gradient descent:

Following is for two class classification(K=2):

$y=P(C|x)=1/(1+\exp(-w^Tx+w_0))$, we use sigmoid function to calculate y

For $j = 0, \dots, d$
    $w_j \leftarrow \text{rand}(-0.01, 0.01)$
Repeat
    For $j = 0, \dots, d$
        $\Delta w_j \leftarrow 0$
    For $t = 1, \dots, N$
        $o \leftarrow 0$
        For $j = 0, \dots, d$
            $o \leftarrow o + w_j x_j^t$
        $y \leftarrow \text{sigmoid}(o)$
        For $j = 0, \dots, d$
            $\Delta w_j \leftarrow \Delta w_j + (r^t - y)x_j^t$
    For $j = 0, \dots, d$
        $w_j \leftarrow w_j + \eta \Delta w_j$
Until convergence

If the classification problem is a multi-class classification(K>2), use following update rule:We use softmax function to calculate y.

$y=\exp(o\_k)/(\Sigma_i \exp(o\_i))$

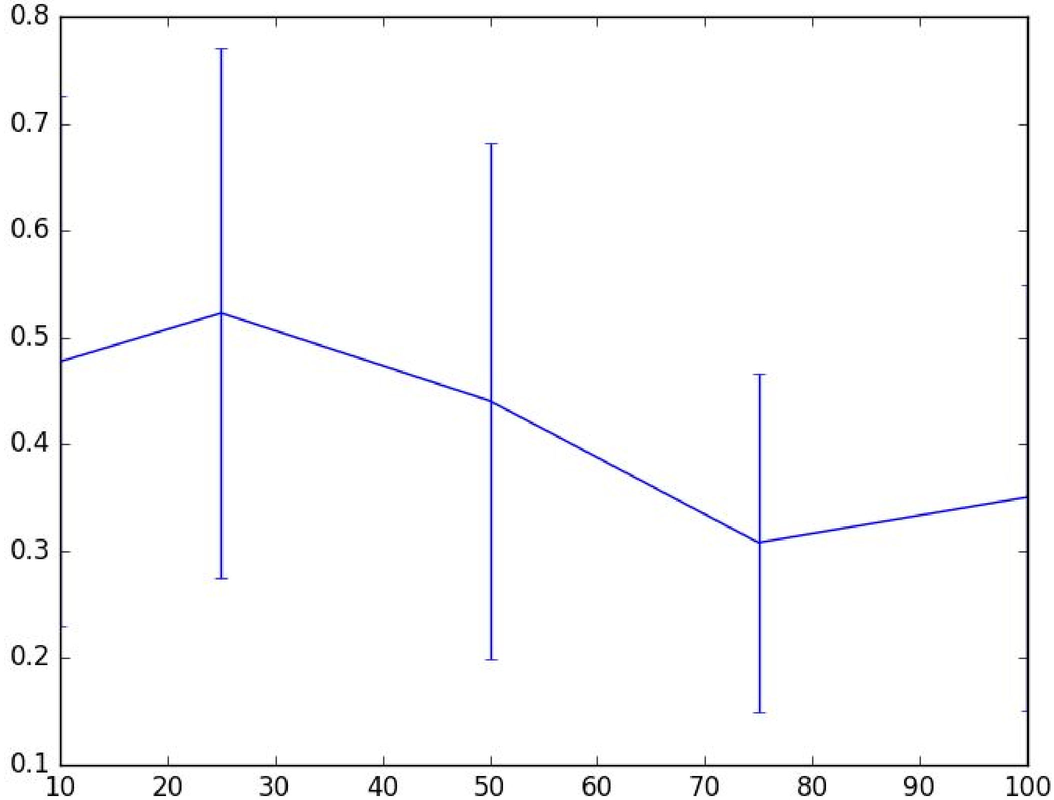
$o\_k=w_k^TX$

```
For i = 1,...,K
    For j = 0,...,d
        w_ij ← rand(−0.01, 0.01)
Repeat
    For i = 1,...,K
        For j = 0,...,d
            Δw_ij ← 0
    For t = 1,...,N
        For i = 1,...,K
            o_i ← 0
            For j = 0,...,d
                o_i ← o_i + w_ij x_j^t
        For i = 1,...,K
            y_i ← exp(o_i) / Σ_k exp(o_k)
        For i = 1,...,K
            For j = 0,...,d
                Δw_ij ← Δw_ij + (r_i^t − y_i) x_j^t
    For i = 1,...,K
        For j = 0,...,d
            w_ij ← w_ij + η Δw_ij
Until convergence
```

Both these two rules came from Introduction to Machine Learning book.
(ii)
Naive bayes with marginal Gaussian distribution:
P(x|C_k)=P(x_1|C_k)*P(x_2|C_k)...*P(x_d|C_k)*P(C_k)

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{D} p(x_i|C_k) = \frac{1}{(2\pi)^{D/2}\left(\prod_{i=1}^{D}\sigma_{ik}\right)} \exp\left\{-\sum_{i=1}^{D}\frac{(x_i-\mu_{ik})^2}{2\sigma_{ik}^2}\right\}$$

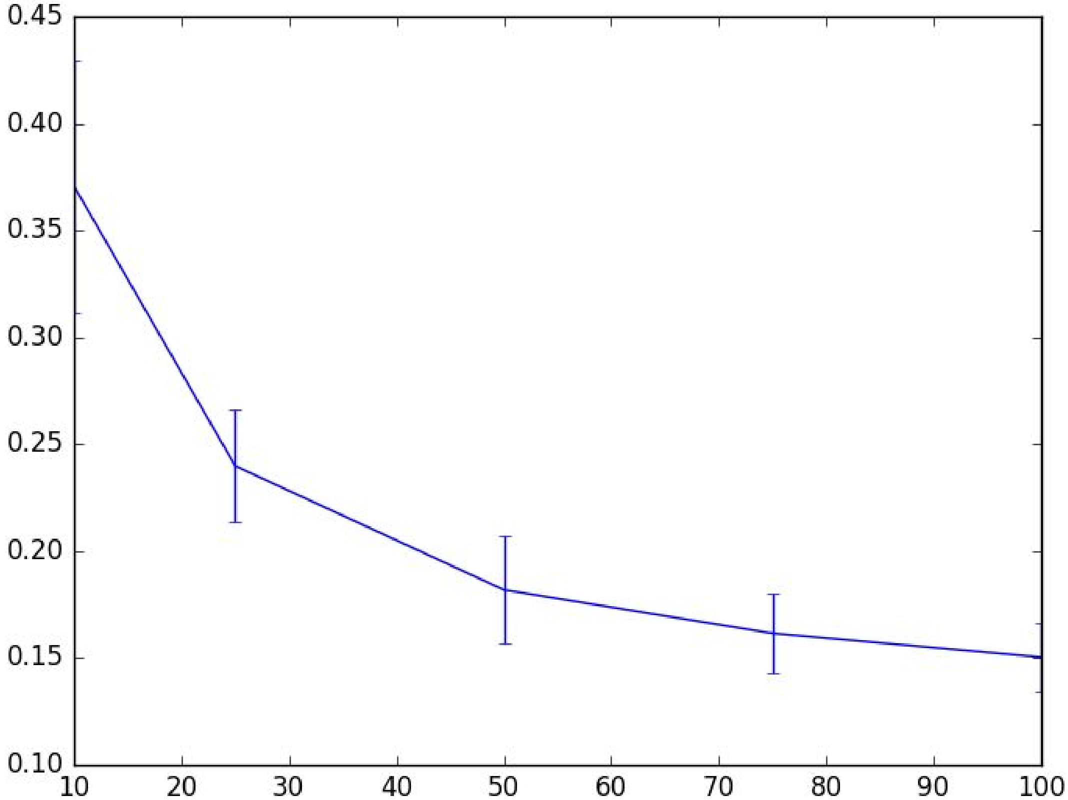
P(C_k)=N_k/N
First, we calculate the sigma, mean and prior probability.
Then we use X to calculate posterior probabilities of different classes and select the largest one as class label
Boston50 dataset:
(i)



Mean_error_rate:
[0.47722772 0.52277228 0.44059406 0.30792079 0.35049505]
Error_matrix:(row index represents num_split and column index is train_percent) :
[[0.54455446 0.45544554 0.45544554 0.45544554 0.54455446]
 [0.10891089 0.89108911 0.89108911 0.10891089 0.10891089]
 [0.85148515 0.14851485 0.14851485 0.14851485 0.14851485]
 [0.22772277 0.77227723 0.77227723 0.22772277 0.22772277]
 [0.08910891 0.91089109 0.08910891 0.08910891 0.08910891]
 [0.47524752 0.52475248 0.52475248 0.52475248 0.47524752]
 [0.57425743 0.42574257 0.42574257 0.42574257 0.57425743]
 [0.61386139 0.38613861 0.38613861 0.38613861 0.61386139]
 [0.78217822 0.21782178 0.21782178 0.21782178 0.21782178]
 [0.5049505   0.4950495   0.4950495   0.4950495   0.5049505]]
Error standard deviation:
[0.24799164 0.24799164 0.24184574 0.15850554 0.19916501]

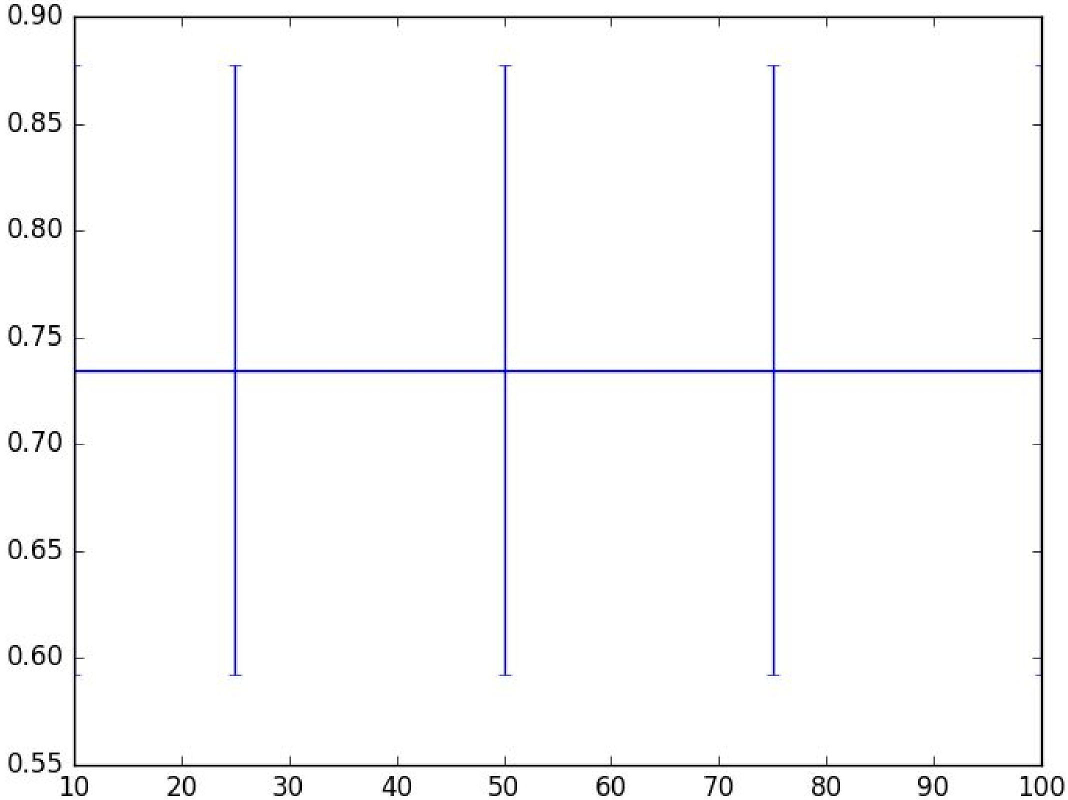
(ii)

(i)



Mean_error_rate:
[0.34653465 0.23861386 0.20792079 0.1960396   0.18613861]
Error_matrix:(row index represents num_split and column index is train_percent)
[[0.4950495   0.14851485 0.27722772 0.16831683 0.10891089]
 [0.5049505   0.05940594 0.20792079 0.10891089 0.04950495]
 [0.17821782 0.27722772 0.18811881 0.16831683 0.13861386]
 [0.38613861 0.34653465 0.26732673 0.3960396   0.3960396]
 [0.48514851 0.24752475 0.21782178 0.26732673 0.27722772]
 [0.14851485 0.22772277 0.0990099   0.13861386 0.14851485]
 [0.45544554 0.27722772 0.24752475 0.14851485 0.18811881]
 [0.13861386 0.27722772 0.16831683 0.12871287 0.13861386]
 [0.4950495   0.24752475 0.21782178 0.26732673 0.27722772]
 [0.17821782 0.27722772 0.18811881 0.16831683 0.13861386]]
Error standard deviation:
[0.15522782 0.07623762 0.04930653 0.08375559 0.09668559]

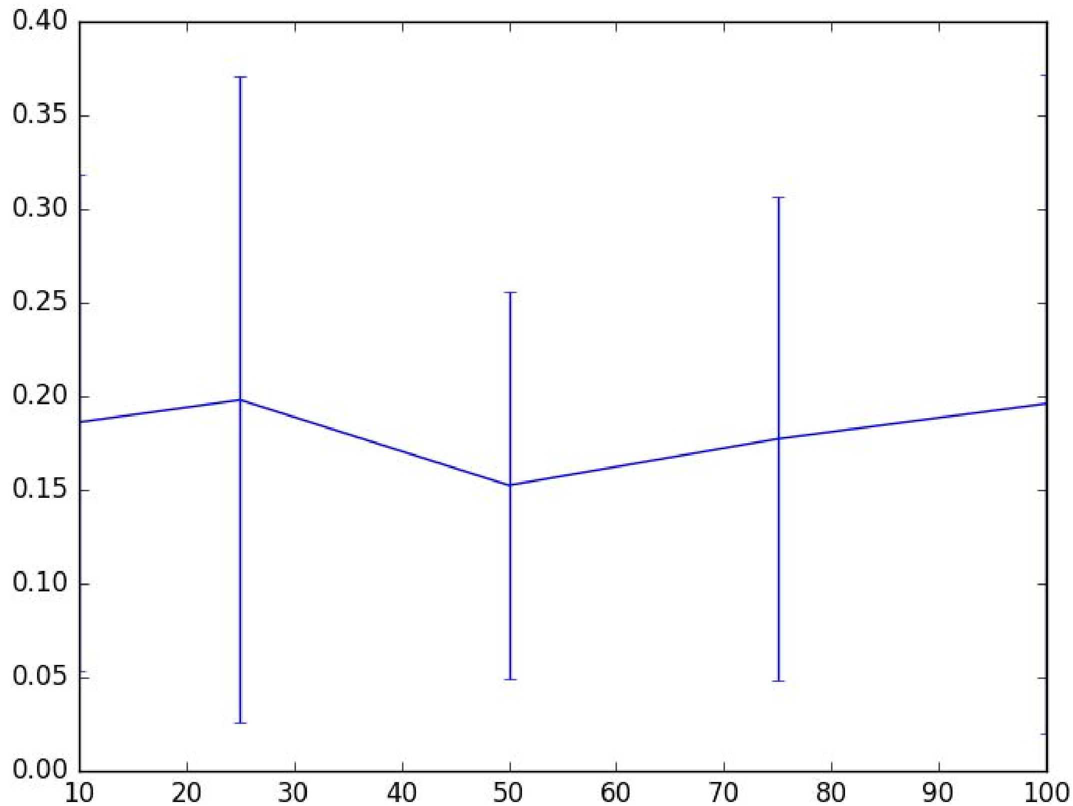
Boston75 dataset:
(i)



Mean_error_rate:
[0.73465347 0.73465347 0.73465347 0.73465347 0.73465347]
Error_matrix:(row index represents num_split and column index is train_percent)
[[0.87128713 0.87128713 0.87128713 0.87128713 0.87128713]
 [0.3960396   0.3960396   0.3960396   0.3960396   0.3960396]
 [0.81188119 0.81188119 0.81188119 0.81188119 0.81188119]
 [0.58415842 0.58415842 0.58415842 0.58415842 0.58415842]
 [0.82178218 0.82178218 0.82178218 0.82178218 0.82178218]
 [0.67326733 0.67326733 0.67326733 0.67326733 0.67326733]
 [0.7029703   0.7029703   0.7029703   0.7029703   0.7029703]
 [0.83168317 0.83168317 0.83168317 0.83168317 0.83168317]
 [0.79207921 0.79207921 0.79207921 0.79207921 0.79207921]
 [0.86138614 0.86138614 0.86138614 0.86138614 0.86138614]]
Error standard deviation:
[0.14250548 0.14250548 0.14250548 0.14250548 0.14250548]

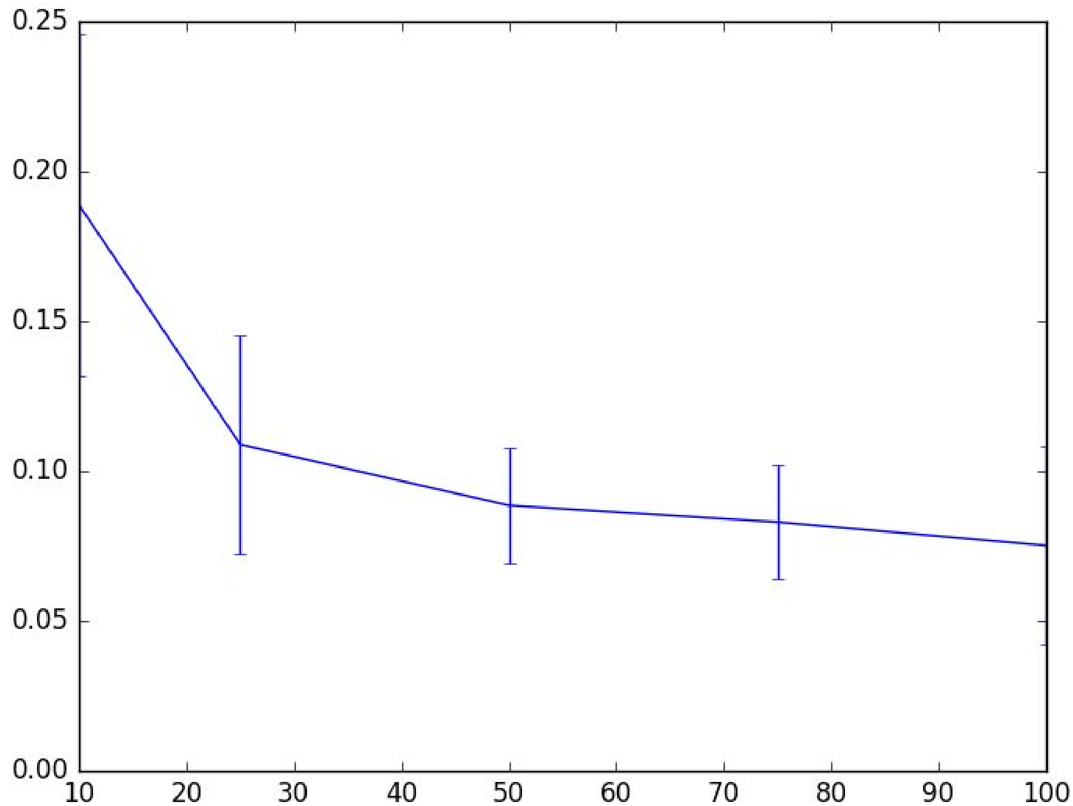
(ii)

(i)

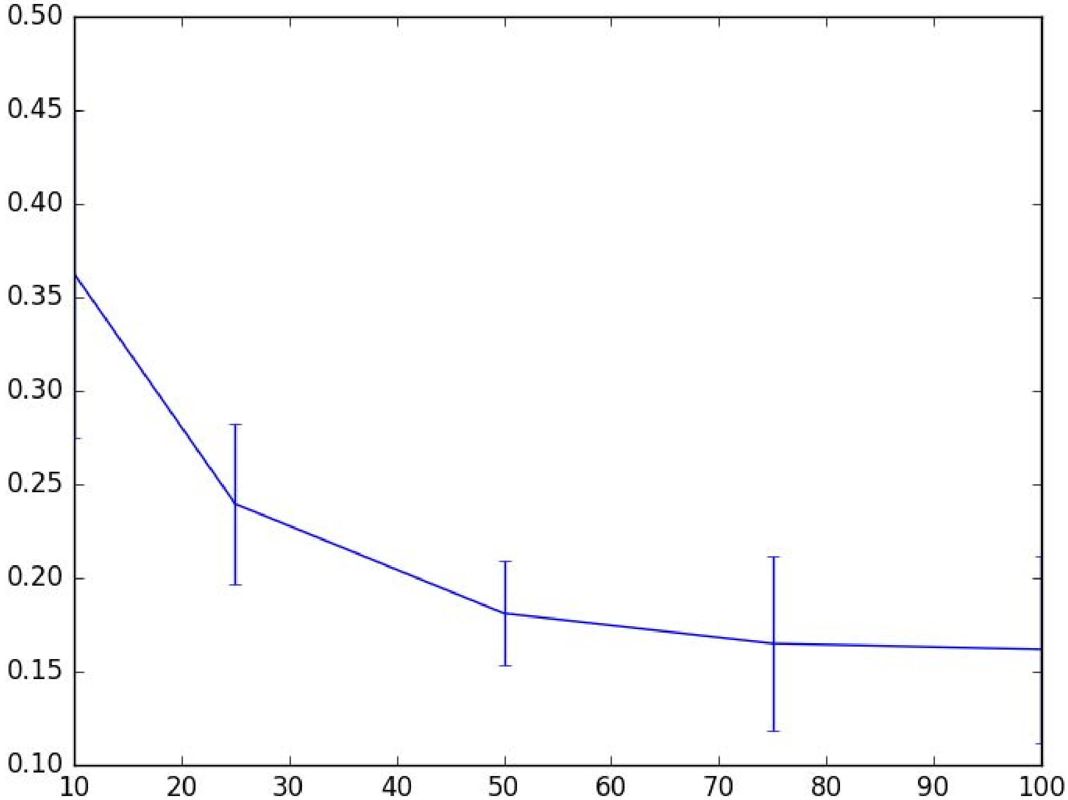


Mean_error_rate:
[0.18613861 0.1980198   0.15247525 0.17722772 0.1960396]
Error_matrix:(row index represents num_split and column index is train_percent)
[[0.23762376 0.10891089 0.11881188 0.0990099   0.08910891]
 [0.3960396   0.35643564 0.25742574 0.21782178 0.30693069]
 [0.03960396 0.03960396 0.04950495 0.06930693 0.03960396]
 [0.02970297 0.02970297 0.02970297 0.04950495 0.02970297]
 [0.30693069 0.41584158 0.30693069 0.41584158 0.48514851]
 [0.35643564 0.21782178 0.14851485 0.16831683 0.20792079]
 [0.14851485 0.15841584 0.17821782 0.16831683 0.16831683]
 [0.23762376 0.54455446 0.31683168 0.41584158 0.52475248]
 [0.06930693 0.06930693 0.06930693 0.07920792 0.06930693]
 [0.03960396 0.03960396 0.04950495 0.08910891 0.03960396]]
Error standard deviation:
[0.13245142 0.17251605 0.10338834 0.12927903 0.17576977]

Digits dataset:
(i)



Mean_error_rate:
[0.18885794 0.10891365 0.08857939 0.08300836 0.07520891]
Error_matrix:(row index represents num_split and column index is train_percent)
[[0.13649025 0.06128134 0.07520891 0.07799443 0.05292479]
 [0.28412256 0.15320334 0.11977716 0.09749304 0.11420613]
 [0.15320334 0.09470752 0.07799443 0.08913649 0.06406685]
 [0.15320334 0.06963788 0.06685237 0.06685237 0.02785515]
 [0.13091922 0.07799443 0.06685237 0.05571031 0.04456825]
 [0.27019499 0.15598886 0.09192201 0.10584958 0.11420613]
 [0.2005571   0.1281337   0.11699164 0.11142061 0.10027855]
 [0.14206128 0.08635097 0.0724234   0.05292479 0.0362117]
 [0.25905292 0.1643454   0.1086351   0.09470752 0.11977716]
 [0.15877437 0.09749304 0.08913649 0.07799443 0.07799443]]
Error standard deviation:
[0.05700174 0.03637098 0.01920996 0.01916953 0.03314646]
(ii)



Mean_error_rate:

```
[ 0.36267409  0.23955432  0.1810585   0.16490251  0.16183844]
```

Error_matrix:(row index represents num_split and column index is train_percent)

```
[[ 0.4735376   0.26462396  0.19777159  0.17827298  0.16155989]
 [ 0.47632312  0.29805014  0.22284123  0.25069638  0.25626741]
 [ 0.32033426  0.22841226  0.18662953  0.17270195  0.14206128]
 [ 0.47075209  0.29805014  0.22005571  0.25069638  0.25626741]
 [ 0.40668524  0.2367688   0.13370474  0.11420613  0.09192201]
 [ 0.23398329  0.16155989  0.15041783  0.14206128  0.1448468 ]
 [ 0.23119777  0.16991643  0.15041783  0.13649025  0.14206128]
 [ 0.32590529  0.23955432  0.1810585   0.1448468   0.13927577]
 [ 0.3454039   0.25069638  0.18662953  0.13370474  0.1448468 ]
 [ 0.34261838  0.24791086  0.1810585   0.12534819  0.13927577]]
```

Error standard deviation:

```
[ 0.08747405  0.0432787   0.02779939  0.04669042  0.0500998 ]
```