ICSI431 Homework 3 Seoyeon Choi

Part 1 [28 pts.] Bayes Classifier. Consider the following dataset, classify the new point: (Age=23, Car=truck) via the full and naive Bayes approach, assuming that the domain of Car is given as sports, vintage, suv, truck.

Full Bayes Approach:

1. Calculate prior probabilities for classes L and H:

$$P(L) = \frac{2}{6}, \quad P(H) = \frac{4}{6}$$

2. Calculate likelihood $P(x|c_i)$ using the smoothing method:

$$P(\text{age} = 23, \text{car} = truck|L) = \frac{0+1}{2+(4\times 4)} = \frac{1}{18}$$

$$P(\text{age} = 23, \text{car} = truck|H) = \frac{0+1}{24+(4\times 4)} = \frac{1}{20}$$

3. Calculate posterior probabilities $P(c|x)$:

$$P(L|\text{age} = 23, \text{car} = truck) = P(\text{age} = 23, \text{car} = truck|L) \times P(L)$$
$$= \left(\frac{2}{6}\right) \times \left(\frac{1}{18}\right) = 0.0185$$

$$P(H|\text{age} = 23, \text{car} = truck) = P(\text{age} = 23, \text{car} = truck|H) \times P(H)$$
$$= \left(\frac{4}{6}\right) \times \left(\frac{1}{20}\right) = 0.0333$$

Naive Bayes Approach:

1. Calculate prior probabilities for classes L and H:

$$P(L) = \frac{2}{6}, \quad P(H) = \frac{4}{6}$$

2. Calculate likelihood $P(x|c_i)$ using normal distribution for the age attribute and the smoothing method for the categorical car attribute:

$$
\begin{aligned}
P(\text{age} = 23|L) &= \left(\frac{1}{\sqrt{2\pi} \cdot 0}\right) \cdot \exp\left(-\frac{(23-25)^2}{2 \cdot 0}\right) = 0 \\
P(\text{age} = 23|H) &= \left(\frac{1}{\sqrt{2\pi} \cdot 10.31}\right) \cdot \exp\left(-\frac{(23-27.5)^2}{2 \cdot 106.25}\right) \\
P(\text{car} = truck|L) &= \frac{0+1}{2+4} = \frac{1}{6} \\
P(\text{car} = truck|H) &= \frac{0+1}{4+4} = \frac{1}{8}
\end{aligned}
$$

3. Calculate posterior probabilities $P(c|x)$:

$$
\begin{aligned}
P(L|\text{age} = 23, \text{car} = truck) &= P(\text{age} = 23, \text{car} = truck|L) \times P(L) \\
&= \left(\frac{2}{6}\right) \times \left(\frac{1}{18}\right) = 0.0185 \\
P(H|\text{age} = 23, \text{car} = truck) &= P(\text{age} = 23, \text{car} = truck|H) \times P(H) \\
&= \left(\frac{4}{6}\right) \times \left(\frac{1}{20}\right) = 0.0333
\end{aligned}
$$

Naive Bayes Approach:

1. Calculate prior probabilities for classes L and H:

$$P(L) = \frac{2}{6}, \quad P(H) = \frac{4}{6}$$

2. Calculate likelihood $P(x|c_i)$ using normal distribution for the age attribute and the smoothing method for the categorical car attribute:

$$\begin{aligned}
P(\text{age} = 23|L) &= \left(\frac{1}{\sqrt{2\pi} \cdot 0}\right) \cdot \exp\left(-\frac{(23-25)^2}{2 \cdot 0}\right) = 0 \\
P(\text{age} = 23|H) &= \left(\frac{1}{\sqrt{2\pi} \cdot 10.31}\right) \cdot \exp\left(-\frac{(23-27.5)^2}{2 \cdot 106.25}\right) \\
P(\text{car} = \textit{truck}|L) &= \frac{0+1}{2+4} = \frac{1}{6} \\
P(\text{car} = \textit{truck}|H) &= \frac{0+1}{4+4} = \frac{1}{8}
\end{aligned}$$

3. Calculate posterior probabilities:

$$\begin{aligned}
P(H|\text{age} = 23, \text{car} = \textit{truck}) &= P(\text{age} = 23|H) \times P(\text{car} = \textit{truck}|H) \times \\
&= 0.035 \times \left(\frac{1}{8}\right) \times \left(\frac{4}{6}\right) = 0.002913 \\
P(L|\text{age} = 23, \text{car} = \textit{truck}) &= P(\text{age} = 23|L) \times P(\text{car} = \textit{truck}|L) \times \\
&= 0 \times \left(\frac{2}{6}\right) = 0
\end{aligned}$$

Therefore, both Full Bayes and Naive Bayes approaches classify the instance (age=23, car=truck) as belonging to class H.

Part 2 – (d)

Run the code on the cancer dataset with different values of ψ and ϵ. Check the change in cross-entropy values across iterations (in the plot) and the average training and testing cross-entropy errors. What do you observe about the losses and number of iterations? What do you conclude?

First, we tried with learning rate $\psi = 0.1$, $\epsilon = 10$ (as stopping criterion), and max epochs = 5. This is the list of w values obtained from logisticRegression SGA with x_{train}, y_{train}, $\psi = 0.1$, $\epsilon = 10$, max epochs = 5.

```
(base) choeseoyeon@MacBook-Pro-3 HW3 % ls
HW3.pdf                 cancer-data-train.csv   logisticRegression.py
cancer-data-test.csv    classifiers.py
(base) choeseoyeon@MacBook-Pro-3 HW3 % python3 logisticRegression.py
w :
 [ 0.00656    0.007865   0.007055   0.0046935  0.005685  -0.006845
 -0.006795   0.00776    0.00811   -0.00644   -0.006175   0.00817
 -0.00652   -0.00559   -0.0048505  0.007095  -0.005125  -0.00763
 -0.005995   0.00827   -0.00611   -0.007335   0.00759   -0.00583
 -0.006675   0.00614   -0.00658   -0.005085  -0.006725  -0.00628
  0.006405   0.006325   0.005555  -0.00701   -0.00552   -0.006635
  0.005655  -0.007145   0.007005   0.00636   -0.005615   0.005485
  0.00897   -0.00718   -0.00662   -0.00595   -0.0061    -0.007135
  0.008885  -0.006815   0.008205   0.00787   -0.006355   0.01113
 -0.005715  -0.00572    0.00713    0.006275   0.005695  -0.00632
 -0.005365   0.007175   0.007055  -0.006845  -0.0047225 -0.00752
 -0.00608    0.006975   0.007665   0.00741    0.00687    0.00891
 -0.00603    0.006215   0.007655   0.009545   0.0075    -0.0073
 -0.00613    0.005185  -0.005855  -0.00532    0.007575   0.00825
 -0.00701    0.007245  -0.00568    0.005105  -0.007575  -0.0079
 -0.00692   -0.005485  -0.005475  -0.00669   -0.005225  -0.005005
  0.00722   -0.00558   -0.00564    0.00719    0.00732   -0.00687
 -0.006925   0.00939   -0.006745  -0.005385  -0.00798   -0.005365
  0.00661   -0.00641   -0.00701    0.00567   -0.00641   -0.00703
 -0.00515   -0.00554   -0.00559   -0.00805   -0.00611   -0.00609
 -0.00725   -0.005405  -0.004979   0.007455  -0.0047635 -0.005735
 -0.00571    0.006825   0.00708   -0.00656   -0.00713   -0.006945
 -0.0047285  0.007545  -0.006885   0.008195   0.008945  -0.00617
 -0.005975  -0.006745   0.00619   -0.00692   -0.00862    0.005995
 -0.006555  -0.00688    0.00749    0.00696    0.01092   -0.00678
```

Figure 1: W values with $\psi = 0.1$, $\epsilon = 10$, max epochs = 5

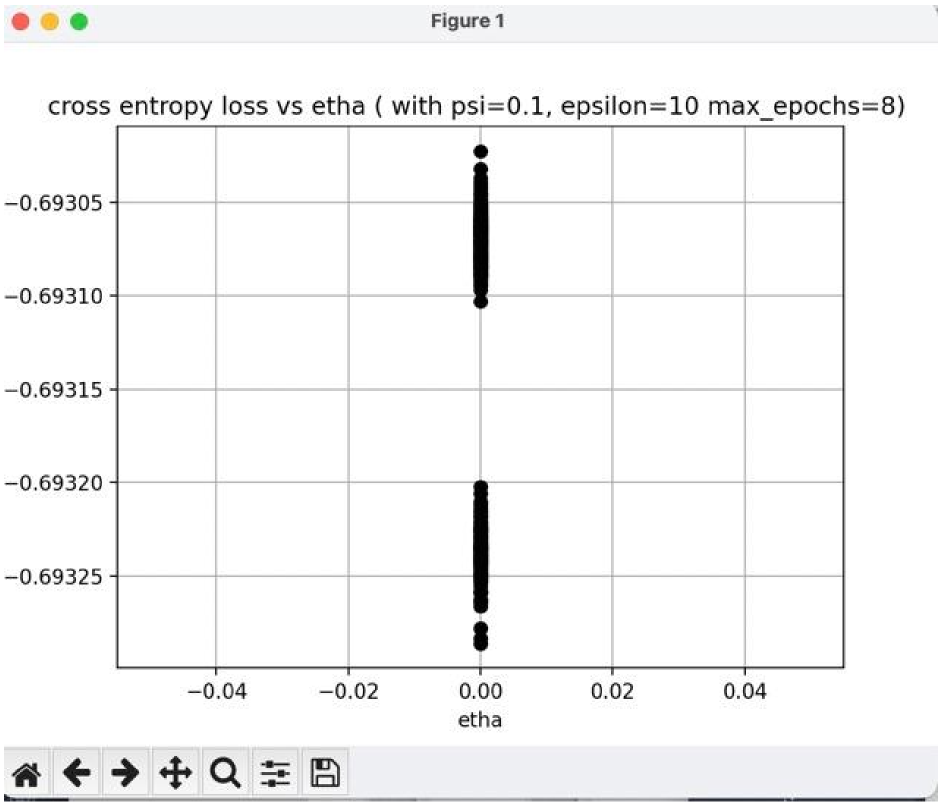
This is the cross-entropy loss returned from the logisticRegression SGA()

method.

HW3 — python3 logisticRegression.py — 127×33

```
  0.00752  ]
cross ent :
 [array([-0.69322899, -0.69324529, -0.69323517, -0.6932057 , -0.69321807,
       -0.69306202, -0.69306264, -0.69324398, -0.69324835, -0.69306705,
       -0.69307035, -0.6932491 , -0.69306606, -0.69307762, -0.69308681,
       -0.69323567, -0.6930834 , -0.69305227, -0.69307258, -0.69325035,
       -0.69307115, -0.69305593, -0.69324186, -0.69307463, -0.69306413,
       -0.69322375, -0.69306531, -0.6930839 , -0.69306351, -0.69306904,
       -0.69322706, -0.69322606, -0.69321645, -0.69305997, -0.69307849,
       -0.69306463, -0.6932177 , -0.69305829, -0.69323455, -0.6932265 ,
       -0.69307731, -0.69321557, -0.6932591 , -0.69305786, -0.69306482,
       -0.69307314, -0.69307128, -0.69305842, -0.69325804, -0.69306239,
       -0.69324954, -0.69324535, -0.69306811, -0.6932861 , -0.69307606,
       -0.693076  , -0.69323611, -0.69322544, -0.69321819, -0.69306854,
       -0.69308042, -0.69323667, -0.69323517, -0.69306202, -0.6930884 ,
       -0.69305363, -0.69307153, -0.69323418, -0.69324279, -0.69323961,
       -0.69323286, -0.69325835, -0.69307215, -0.69322469, -0.69324267,
       -0.69326629, -0.69324073, -0.69305637, -0.69307091, -0.69321183,
       -0.69307432, -0.69308097, -0.69324167, -0.6932501 , -0.69305997,
       -0.69323755, -0.6930765 , -0.69321083, -0.69305295, -0.69304891,
       -0.69306109, -0.69307892, -0.69307905, -0.69306395, -0.69308216,
       -0.69308489, -0.69323724, -0.69307774, -0.693077  , -0.69323686,
       -0.69323848, -0.69306171, -0.69306103, -0.69326435, -0.69306326,
       -0.69308017, -0.69304792, -0.69308042, -0.69322962, -0.69306743,
       -0.69305997, -0.69321788, -0.69306743, -0.69305972, -0.69308309,
       -0.69307824, -0.69307762, -0.69304705, -0.69307115, -0.6930714 ,
       -0.69305699, -0.69307992, -0.69308521, -0.69324017, -0.69308789,
       -0.69307582, -0.69307613, -0.6932323 , -0.69323549, -0.69306556,
       -0.69305848, -0.69306078, -0.69308833, -0.69324129, -0.69306152,
       -0.69324941, -0.69325879, -0.69307041, -0.69307283, -0.69306326,
       -0.69322437, -0.69306109, -0.69303997, -0.69322194, -0.69306562,
       -0.69306159, -0.69324061, -0.69323399, -0.69328348, -0.69306283,
       -0.69307625, -0.69323898, -0.69306314, -0.69304513, -0.69323324,
```

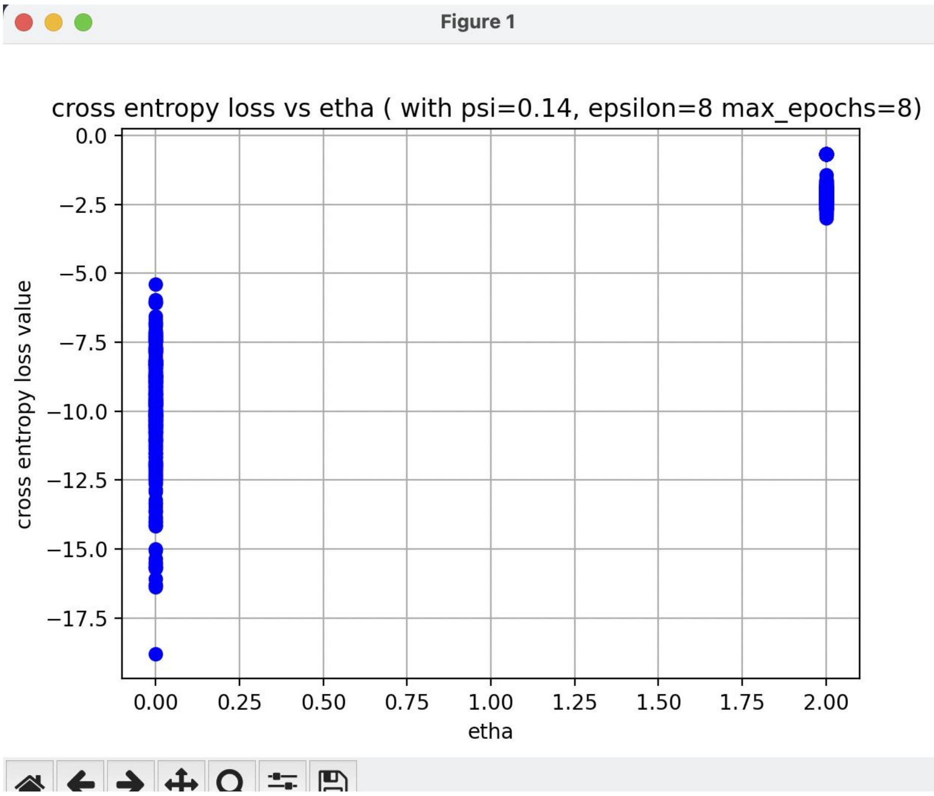
This plot shows the cross-entropy loss vs. η (with ψ = 0.1, ϵ = 10, max epochs = 5).



These are the average cross-entropy errors for training and testing data.

```
Terminal  Shell  Edit  View  Window  Help
HW3 — python3 logisticRegression.py — 127×33
Average cross-entropy error for training data: [[-0.69322899 -0.69324529 -0.69323517 ... -0.69305898 -0.69325104
  -0.69324098]
 [-0.70626752 -0.70937384 -0.70742645 ... -0.6836559  -0.7105079
  -0.70853666]
 [-1.9454793  -2.30717482 -2.08165776 ... -2.56840209 -2.43639884
  -2.21076811]
 ...
 [-2.32534568 -2.77184233 -2.49394527 ... -3.68843952 -2.93049521
  -2.65321942]
 [-0.69319805 -0.69320818 -0.69320189 ... -0.69309229 -0.69321175
  -0.6932055 ]
 [-0.69314559 -0.69314528 -0.69314547 ... -0.69314892 -0.69314517
  -0.69314536]]
Average cross-entropy error for testing data: [[-0.69393976 -0.69409871 -0.694      ... -0.69230266 -0.69415484
  -0.69405665]
 [-0.69671895 -0.69771584 -0.69708591 ... -0.69199042 -0.69808984
  -0.69744307]
 [-0.70309952 -0.70559429 -0.70402576 ... -0.68725769 -0.70651424
  -0.7049182 ]
 ...
 [-0.69307645 -0.6930624  -0.69307112 ... -0.69322385 -0.69305745
  -0.69306611]
 [-0.69332211 -0.69335701 -0.69333534 ... -0.69295912 -0.69336933
  -0.69334778]
 [-0.69320259 -0.69321362 -0.69320677 ... -0.69308739 -0.69321751
  -0.6932107 ]]
Finished First Task.
```

Next, I tested with different values of ψ and ϵ. This time, I set $\psi = 0.14$ and $\epsilon = 8$. This plot shows the cross-entropy loss value vs. η.



These are the average cross-entropy errors for training and testing data.

```
Average cross-entropy error for training data: [[          nan           nan           nan ... -87.80308916           nan
            nan]
 [          nan           nan           nan ...           nan           nan
            nan]
 [ -0.67656645  -0.67887014  -0.6736766  ...  -2.0374551   -0.67448763
   -0.67709702]
 ...
 [ -0.69760368  -0.69806751  -0.69616822 ...  -0.67213281  -0.69698093
   -0.69772247]
 [ -2.10424752  -2.30147636  -1.50666874 ... -17.44275365  -1.84158759
   -2.15466716]
 [          nan           nan           nan ...           nan           nan
            nan]]
Average cross-entropy error for testing data: [[ -1.20684096  -1.28548001  -0.98588477 ...  -4.2012198   -1.10633593
   -1.2266972 ]
 [          nan           nan  -8.70937962 ... -76.16672294           nan
            nan]
 [          nan           nan           nan ... -94.58568574           nan
            nan]
 ...
 [ -0.68157222  -0.68111604  -0.68382693 ...  -1.05804028  -0.68239063
   -0.6814431 ]
 [ -0.76595776  -0.77628267  -0.73704828 ...  -1.32111109  -0.75283492
   -0.76855793]
 [ -0.70974782  -0.71168305  -0.70399339 ...  -0.70626976  -0.70720696
   -0.71023998]]
(base) choeseoyeon@MacBook-Pro-3 HW3 %
```

In conclusion, I didn't change the max epochs value, so the number of iterations for SGA was the same. The only thing I changed was the learning rate and epsilon for the stopping criterion. I increased the learning rate ψ from 0.1 to 0.14 and decreased epsilon from 10 to 8. As a result, we can see from the attached plots and the average cross-entropy error for training and testing data that increasing the learning step led to higher cross-entropy loss. This indicates that a learning rate that is too small is not optimal since it moves very slowly and requires more computation. However, increasing the learning rate from 0.1 to 0.14 was not a good choice, as it moved larger steps and increased the cross-entropy loss. Also, we can see that the average cross-entropy error value for training data increased up to –7.7906, and the average cross-entropy error for testing data also slightly increased.

Therefore, with a higher learning rate (ψ), the algorithm takes larger steps towards finding the optimal point and may converge faster, but this could lead to higher cross-entropy loss. Decreasing the stopping criterion (ϵ) means that I changed the norm of the weight vector to be smaller, $\|\tilde{W}^{(t+1)} - \tilde{W}^{t}\| < \epsilon$, but this could result in fewer iterations for convergence. Therefore, to reduce cross-entropy loss, it is important to find a learning rate and stopping criterion that are neither too small nor too large.

Part 3 – (a)

Discuss your observations. Is a smaller or larger margin better for this dataset? (Need to explain which C values are likely to produce smaller versus larger values and then which end up being better in cross-validation.)

In linear SVM, there is a trade-off between maximizing margin width vs minimizing classification epsilon error on the training data. For example, if C is small, this indicates that we care more (give more weight) to have as large as possible margin width rather than focusing on minimizing classification error. With smaller C value and larger margin width, this model will be more generalized to unseen data (such as testing data) and have lower variance (robust).

If C value is larger, this indicates that we care more (give more weight) to minimizing classification error, and this results in reducing margin width. With larger C value, we can reduce classification error, but since we end up with smaller margin width, this model may poorly work on unseen data such as testing data (higher variance and not a generalized model).

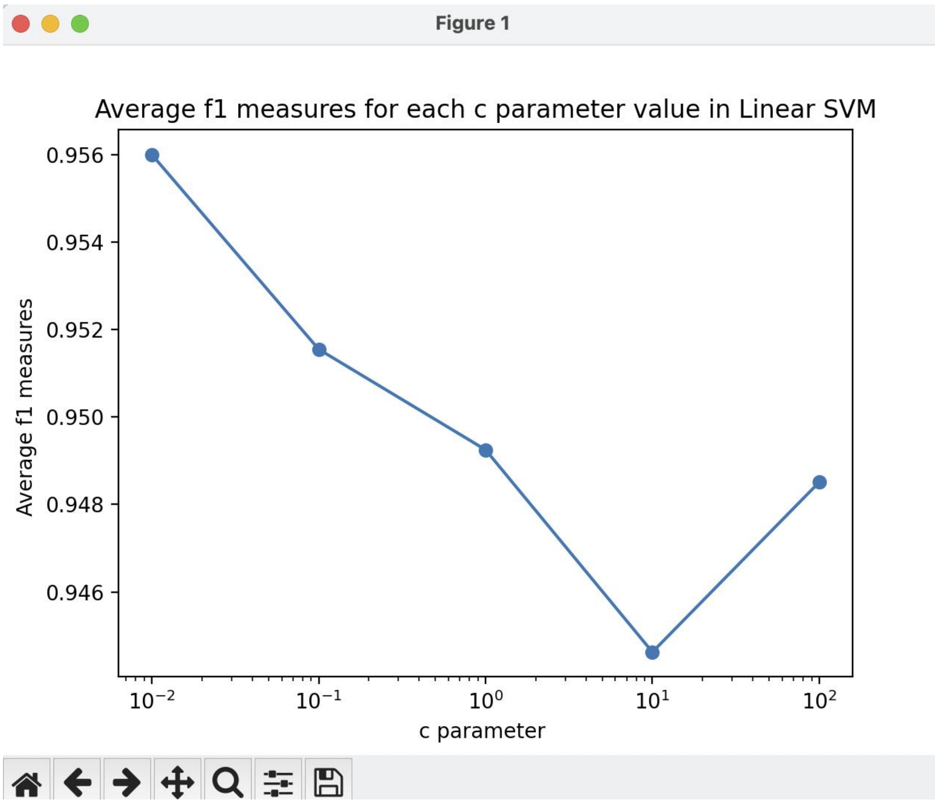
Based on the implemented SVM function for Part 3 – (a), the best C value from the linear SVM was 0.01, which gives the largest margin width, and corresponding C = 0.01 had an average F–measure of around 0.956. Therefore, the smallest C value with the largest margin width was better to

use.

```
HW3 — python3 classifiers.py — 132×55
Last login: Fri Apr 26 14:04:47 on ttys006
[3] 3317
(base) choeseoyeon@MacBook-Pro-3 HW3 % python3 classifiers.py
None
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R algor
ithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R algor
ithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R algor
ithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
```

HW3 — python3 classifiers.py — 116×26

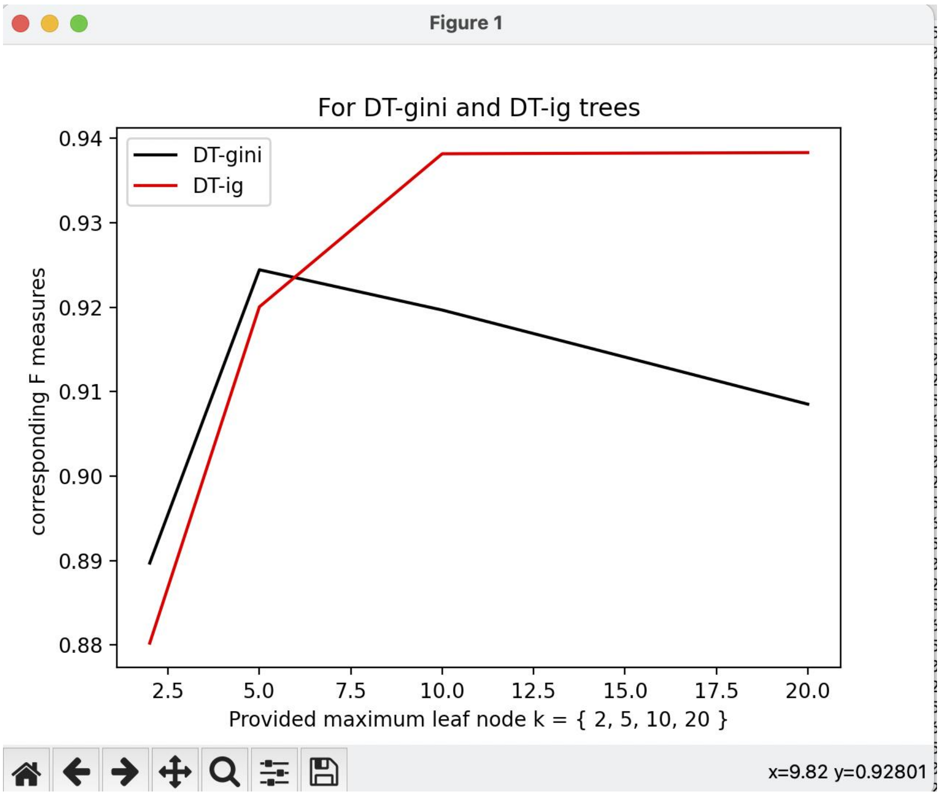
```
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
```



This is the linear SVM Plot.

Part 3 – (b)

Discuss your observations from the figures. Does a larger tree mean a better F–measure? Which criterion is better?



This is the plot for DT–gini and DT–ig tree. The x–axis represents the maximum leaf node *k*, and the y–axis represents the corresponding F–measures.

For DT–gini: F–measure started increasing from maximum leaf node size $k = 2$ to 5, and after $k = 5$, F–measure started to decrease (this indicates overfitting). For DT–ig: F–measure started to increase from maximum leaf node size k = 2 to around 11, after k = 11, F-measure maintained around 0.94.

Does a larger tree mean a better F-measure value?

No. Increasing maximum leaf node size k increases the F-measure value at the beginning, but too much big size k causes overfitting and decreases the F-measure value as we see in the DT-gini plot.

Which criterion is better?

It depends on various factors and dataset characteristics, but based on the output that I got, the output plot of DT-gini and DT-ig shows that DT-ig was able to achieve a higher F-measure value at k = 20 compared to DT-gini, so choosing DT-gini could be a better criterion since it had a higher F-measure. On the side note, it is also important to consider generalizability, robustness, and computational efficiency.

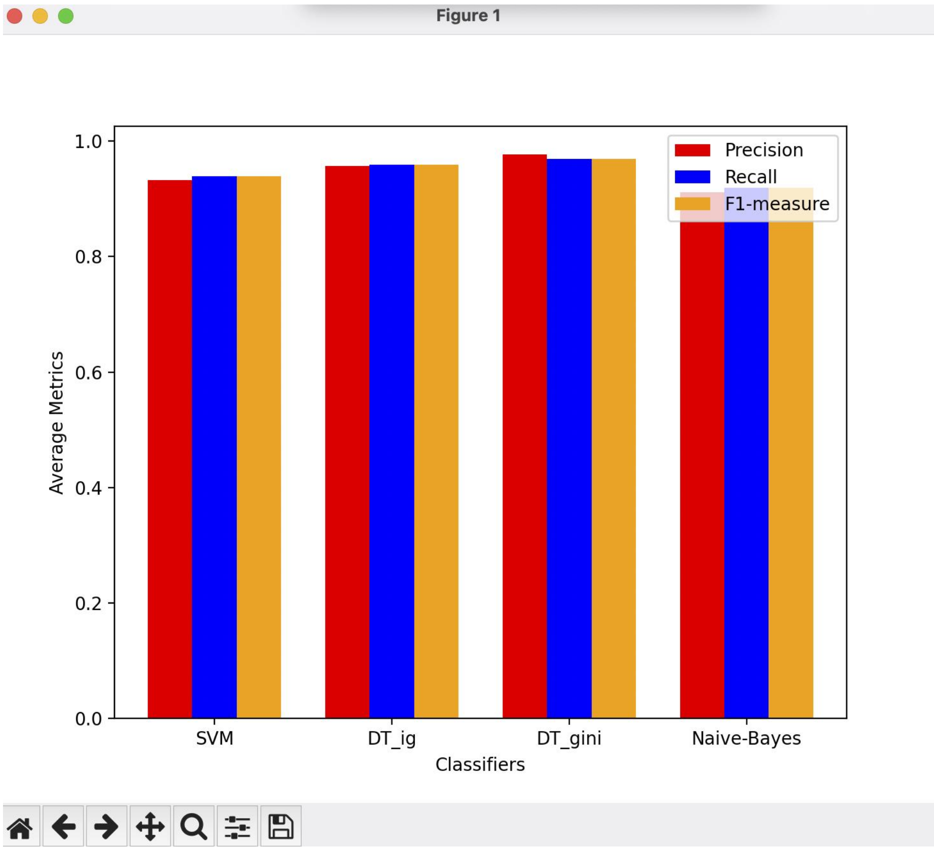
HW3 — python3 classifiers.py — 116×40

```
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697
c :  0.01  average f1 measure value :  0.9560028022421697
c :  0.1  average f1 measure value :  0.9515503116762674
c :  1  average f1 measure value :  0.9492508815920256
c :  10  average f1 measure value :  0.9446236976982847
c :  100  average f1 measure value :  0.9485107016562948
best c value from linear SVM part(a) :  0.01
corresponding average f measure value :  0.9560028022421697

Start part3 - b task

Best size of tree for DT-ig: 20
Corresponding average f1 measure for DT-ig: 0.9186032006776454
Best size of tree for DT-gini: 20
Corresponding average f1 measure for DT-gini: 0.9145582482104638
```

This is best size of the trees for DT-ig and DT-gini from part(b). I will use this k = 20 values in part c.

Part 3 – (c)

Discuss your findings. Which are the best classifiers when you consider the different metrics? Is there a single winner for this dataset?



This is output box plot, where red color represent precision, blue color represent recall, and yellow care represent f1–measure values. Total there are 4 box plot, each of thme is SVM , DT–ig, DT–gini, and Naïve Bayes classifier.

```
HW3 — python3 classifiers.py — 121×55
(base) choeseoyeon@MacBook-Pro-3 HW3 % code .
(base) choeseoyeon@MacBook-Pro-3 HW3 % python3 classifiers.py
None

Start part3 - c task

Size of X_train: (468, 30)
Size of y_train: (468,)
Size of X_test: (99, 30)
Size of y_test: (99,)
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature
 names, but SVC was fitted with feature names
  warnings.warn(
got svm_pred :  ['B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'M'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B'
 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'B']
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature
 names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
got DT_gini_pred :  ['B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B'
 'B' 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B']
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature
 names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
got DT_ig_pred :  ['B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'B']
/Users/choeseoyeon/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature
 names, but GaussianNB was fitted with feature names
  warnings.warn(
got naiveBayes_pred :  ['B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M']
svm_confusin matrix :  [[58  4]
 [ 2 35]]
DT_gini_conf_matrix :  [[60  2]
 [ 2 35]]
DT_ig_conf_matrix :  [[62  0]
 [ 3 34]]
naiveBayes_conf_matrix:  [[57  5]
 [ 3 34]]
```

This is output from the terminal. As we can see, there ar list of predicted SVM value, predicted DT-gini values, predicted DT-ig values, and predicted naive Bayes values.

Based on output box plots all four classifier seems to perform similarly since all of them has similar precision, recall, and f-measure values. In this case, DT-gini has slightly higher precision, recall, f measure value. So If I have to pick only on best classifier for cancer data set, DT-gini classifier is best way to use