CSC411: Project 3

Due on Monday, March 19, 2018

Yufeng Li, Yining Lin

March 19, 2018

Part 1

* Using Python 3.6

Dataset description

The dataset contains 1968 real news headlines and 1298 fake news headlines. All headlines have been cleaned up by removing special characters, words from fake news that are not a part of the headline, and restricting real news headlines to those after October 2016 containing the word trump. This preparation is done by professor. By counting the appearance of word in real and fake news, it might predicted whether the headline is real or not, here are examples:

```
all_words["donald"]
{'real': 829.0, 'fake': 228.0}
all_words['trump']
{'real': 1744.0, 'fake': 1328.0}
all_words['hillary']
{'real': 24.0, 'fake': 150.0}
```

The result shows that the appearances of *trump* and *donald* in real news is more than the appearances in fake news, which means this word is useful for determining the headline is more likely belong to a true news. The appearances of *hillary* indicates that the headline is likely belong to a fake news.

Part 2

We tried several combination of M and P by tuning their value in this part:

```
def get_prob_words_given_C(words_counts, C, num_C, m, p):
    xi_c = {}
    for word, count in words_counts.items():
        xi_c[word] = (min(count[C], num_C) + m * p) / (num_C + m)

    return xi_c
```

The result shows below:

```
****************************
# of set:  1
M and P:  (1.0, 0.05)
Performance:  0.863265306122
****************************
# of set:  2
M and P:  (1.0, 0.1)
Performance:  0.871428571429
****************************
# of set:  3
M and P:  (2.0, 0.05)
Performance:  0.871428571429
****************************
# of set:  4
M and P:  (2.0, 0.1)
Performance:  0.881632653061
****************************
# of set:  5
M and P:  (3.0, 0.05)
Performance:  0.875510204082
****************************
# of set:  6
M and P:  (3.0, 0.1)
Performance:  0.871428571429
```

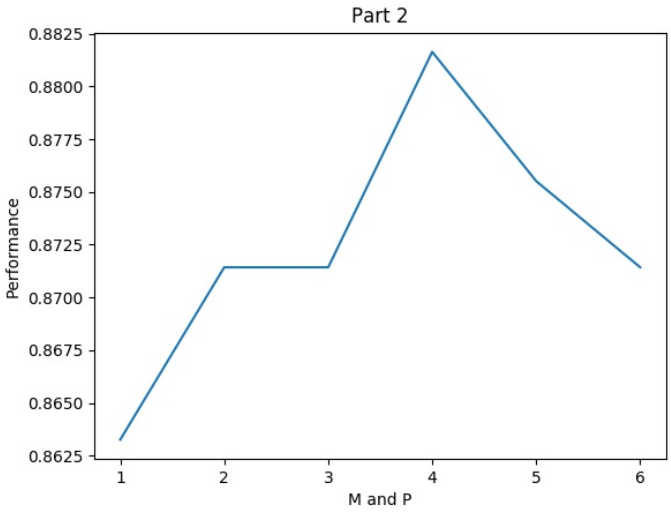

Figure 1: Performance on validation set vs M and P

According to the result, set_6 and set_8 has the best performance on validation set. We decide to use the value from set 6 in rest of the part.
To deal with underflow, we simply add up all the $log(smallnumber)$ by this function:

```
def get_log_sums(small_nums):
    log_sum = 0
    for small_num in small_nums:
        log_sum += np.log(small_num)
    return log_sum
```

Performance of Training set: 0.968517708789
Performance of Test set: 0.848670756646

Part 3

a)

For words whose presence most strongly predicts that the news is real/fake, we need to use this formula:

$$P(Condition|word) = \frac{P(word|Condition)P(Condition)}{P(word)}$$

For words whose absence most strongy predicts that the news is real/fake, we use this formula:

$$P(Condition|not\ word) = P(Condition|all\ words) - P(Condition|the\ word)$$

By applying to the formula above, we have all the probabilities storing in four dictionaries regarding to each word. The result showing below are the top 10 words with highest probabilities in each dictionary.
List the 10 words whose presence most strongly predicts that the news is real.

$$['trump','donald','to','us','on','trumps','in','of','for','says']$$

List the 10 words whose absence most strongly predicts that the news is real.

$$['ron','neocons','watch','shameless','instantly','regrets','fleeing','dreams','secrecy','faked']$$

List the 10 words whose presence most strongly predicts that the news is fake.

$$['trump','the','to','in','donald','for','of','a','and','on']$$

List the 10 words whose absence most strongly predicts that the news is fake.

$$['glass','snl','skit','korea','awkward','handshakes','g20','agenda','scouts','aides']$$

Influence of Presence vs Absence
We compare the performance of presence and absence word by swapping the probabilities of top 10 presence word and top 10 absence word for both real and fake. Here is the result:

```
presence->absence:
training accuracy: 0.881504153913
validating accuracy: 0.728571428571
testing accuracy: 0.760736196319
```

Replace all the probabilities of top 10 absence word to top 10 presence word.

```
absence->presence:
training accuracy: 0.908613904679
validating accuracy: 0.787755102041
testing accuracy: 0.775051124744
```

The original performance is:

```
naive bayes summary:
training accuracy: 0.964145168343
validating accuracy: 0.881632653061
testing accuracy: 0.852760736196
```

We can see that when presence top words becomes absence that performance drop is greater than that of absence top word becomes presence. So presence words have a stronger say in the prediction.

b)

After removing stop words from dictionaries, we get the result below:
10 non-stopwords that most strongly predict that the news is real.

$$['trump','donald','trumps','says','north','election','korea','clinton','ban','russia']$$

10 non-stopwords that most strongly predict that the news is fake.

$$['trump','donald','hillary','clinton','election','just','new','america','president','obama']$$

c)

Stop words such as *about* and *above* should be count as neutral words, which will not contribute to classifying whether the headline is true or not.

Part 4

By trying different combination of α and λ(i.e. the regularization factor), we find out that the best λ should be $\frac{1.0}{Number\ of\ samples}$. The value of α is a common value for regularization.
We choose to use *pytorch* to construct and train the model, and using L2 regulation to avoid overfitting because we derived test using L1,L2 regularization and the performance of testing set is:
L2 Regularization accuracy: 0.828220858896
L1 Regularization accuracy: 0.820040899796
It is obvious that we'd better use L2 regularization.

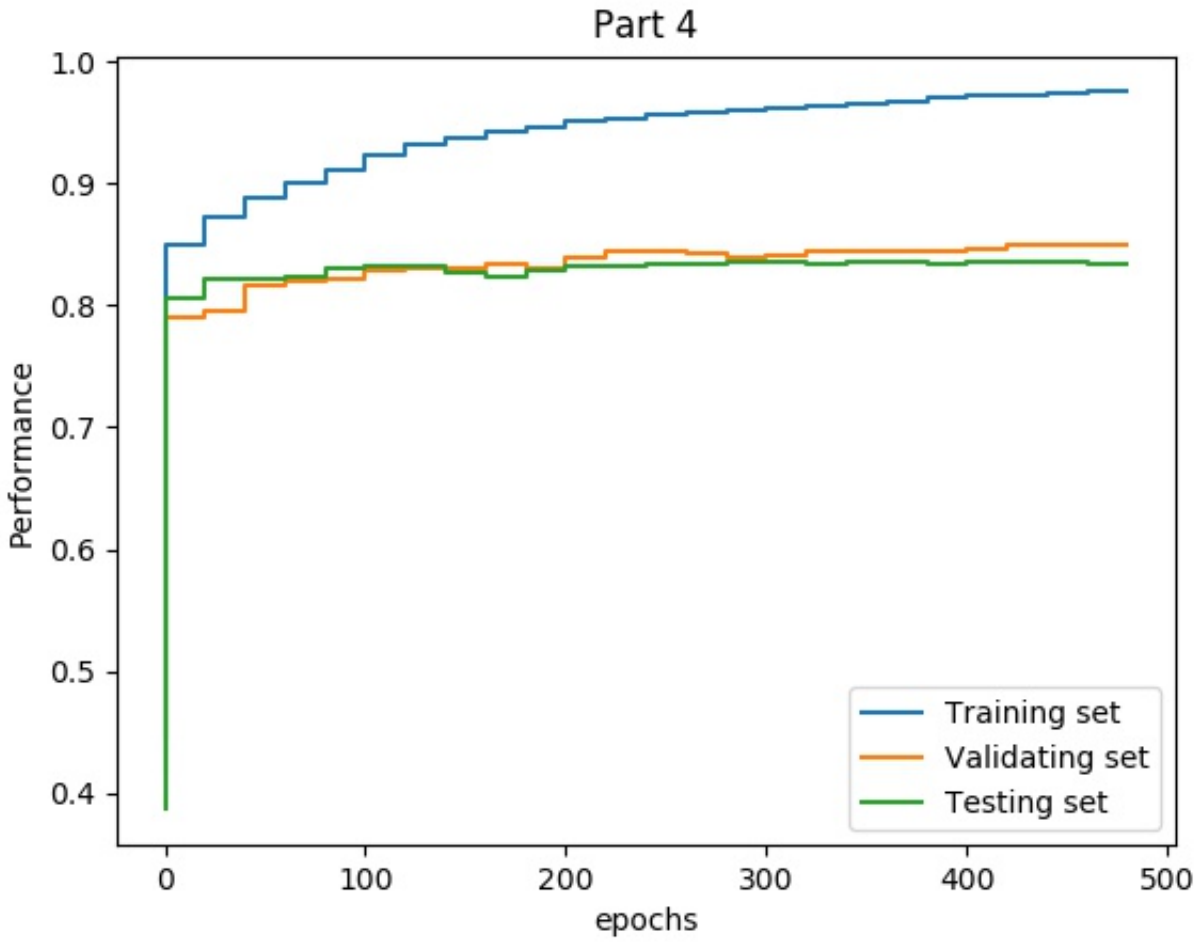

Figure 2: Epochs VS Performance

By using the values below, we get the curve in $Figure2$
$\alpha : 1e-2$
$\lambda : \frac{1.0}{Number\ of\ samples}$
L2 regularization(i.e. train_l2() in logistic.py)

Part 5

Naive Bayes
$\theta_0 : P(C)$
$\theta_i : 1$
$I(x) : \frac{count(x_i=1,C)+mp}{count(C)+m}$

Logistic Regression
θ_0: bias
θ_i: weight of each word
$\mathrm{I(x)}=\begin{cases}1, & if\ word\ appears\\ 0, & otherwise\end{cases}$

Part 6

We use one hot encoding to indicate real$(1,0)$ and fake$(0,1)$, thus every θ is a vector with length equal to 2.

a)

For Real Label
Top 10 positive θ

$$['trumps','turnbull','trump','us','says','donald','korea','debate','ban','north']$$

Top 10 negative θ

$$['hillary','breaking','watch','just','america','victory','new','are','they','the']$$

For Fake Label
Top 10 positive θ

$$['hillary','breaking','watch','just','america','victory','new','are','they','the']$$

Top 10 negative θ

$$['trumps','turnbull','trump','us','says','donald','korea','debate','ban','north']$$

Comparing list of word between part3a and part6a
The list in part3a has more *STOPWORD* such as *to* and *on* comparing to the list in part6a. There are some common words in both list such as *trump* and *says*.

b)

After removing all stop words
For Real Label
Top 10 positive θ

$$['trumps','turnbull','trump','says','donald','korea','debate','ban','north','comey']$$

Top 10 negative θ

$$['hillary','breaking','watch','just','america','victory','new','voter','star','supporter']$$

For Fake Label
Top 10 positive θ

$$['hillary','breaking','watch','just','america','victory','new','star','voter','supporter']$$

Top 10 negative θ

$$['trumps','turnbull','trump','says','donald','korea','debate','ban','north','comey']$$

Comparing list of word between part3b and part6b
There are some common words in both list such as *donald* and *hillary*. By comparing the rank of the same word in different list, it shows that the two algorithm using different way to weight them.

c)

We don't compare magnitude in regression. For example, assume there is a big magnitude and a small magnitude, they contribute the same for the outcome. It is incorrect to use importance to conclude the comparison. For Logistic Regression, we only focus on whether something happen or not, not their time of occurrence. Because the coefficient is useful to formulate into formulas, thus we can plug the input to get prediction.

Part 7

a)

Show the relationship between the *max_depth* of the tree, and the training / validation accuracy

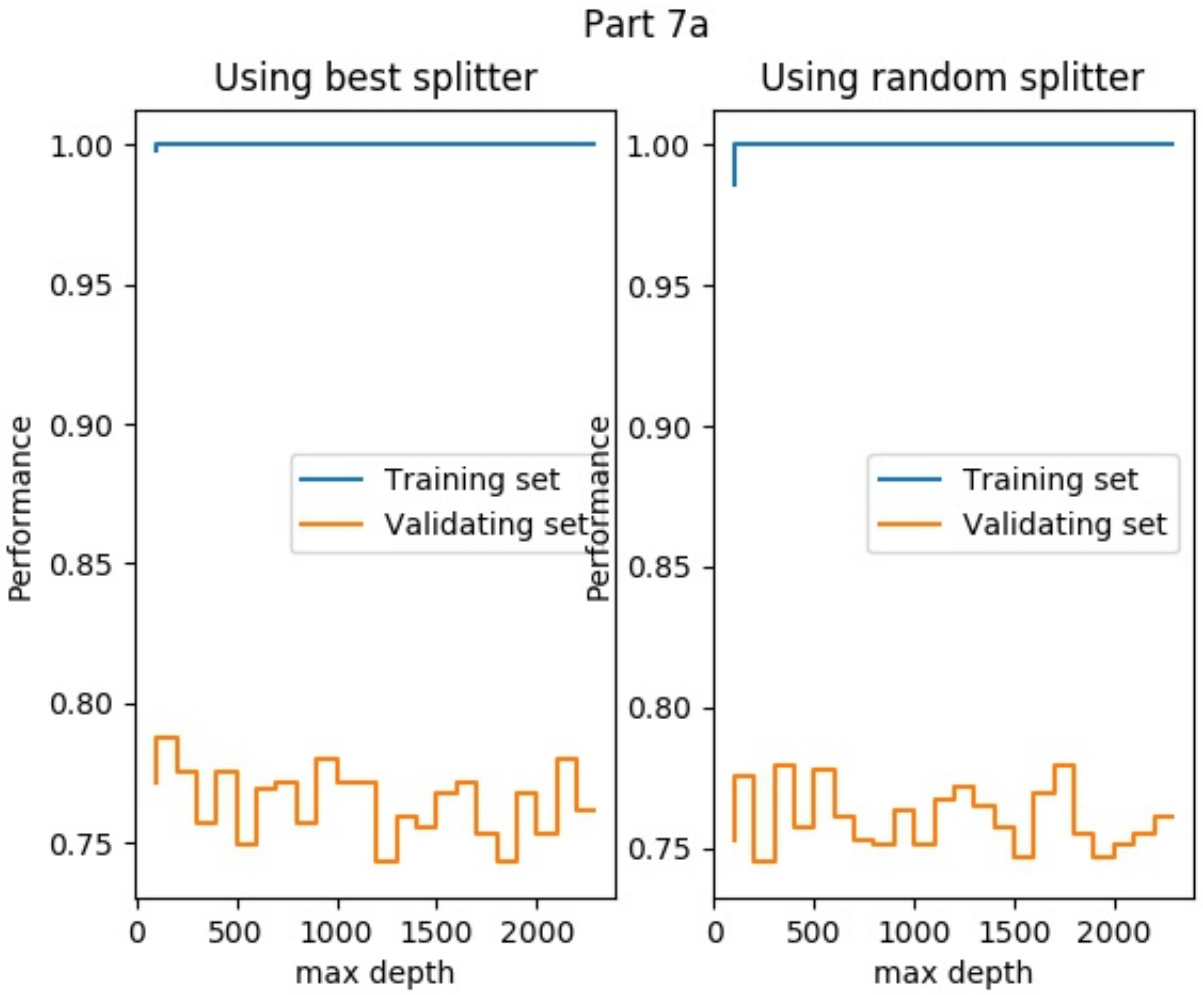

Figure 3: Maxdepth VS Performance(generated using >20 different max depths)

Here I applied different splitter to see their performance, overall best splitter has a better outcome

```
best maxdepth for best splitter: 200
its validating performance: 0.787755102041
best maxdepth for random splitter: 400
its validating performance: 0.779591836735
```

Also, it appears that best splitter acquires smaller max depth thus a faster training.
Finally, the code I'm using to train decision tree:

```
tree.DecisionTreeClassifier(splitter='best', presort=True, max_depth=200,/
criterion='entropy')
```

According to sklearn documents, presort can speed up the training and we are using entropy here to train.

b)

The code I use to generate visualization:

```
import graphviz
from subprocess import call
clf = train(train_x, train_y, 200, 'best')
tree.export_graphviz(clf, out_file='tree.dot', feature_names=words_order, max_depth=3,/
class_names=['fake', 'real'])
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png'])
```

And I extracted the top three levels of the tree:

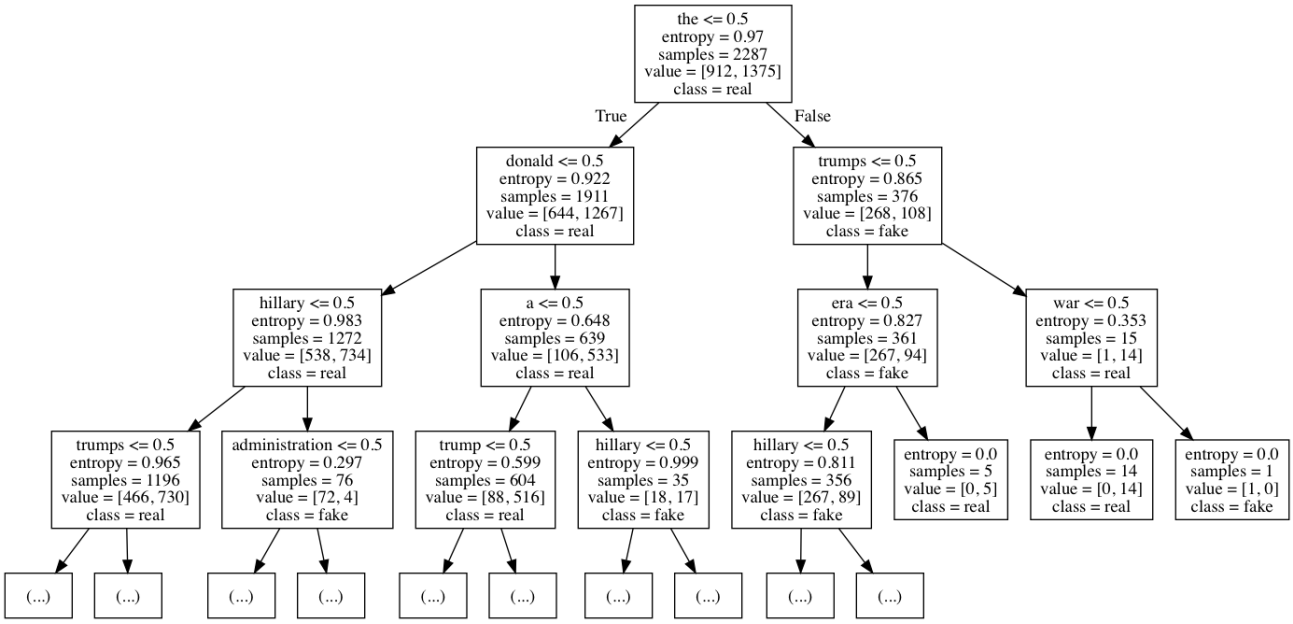

Figure 4: Top 3 level of trained decision tree

I noticed that the most important features was:

$$['the','donald','trump','hillary','a','era','war','trumps','administration']$$

It shares some similarity with both of them. Some of the most importance feature appeared in top 10 positive/negative θ of logistic regression or most strong presence/absence of naive bayes. This indicates that those words are very important to determine in these three methods of classification.

c)

The performance summaries of three classifications are:

```
naive bayes summary:
training accuracy: 0.964145168343
validating accuracy: 0.881632653061
testing accuracy: 0.852760736196

logistic regression summary:
training accuracy: 0.946217752514
```

```
validating accuracy: 0.834693877551
testing accuracy: 0.826175869121

decision tree summary:
training accuracy: 1.0
validating accuracy: 0.781632653061
testing accuracy: 0.768916155419
```

Overall, I would say naive bayes' performance is relatively the best and train faster than logistic regression but a little bit slower than decision tree, presumably because I heavily used dictionary in naive bayes training which maybe slower than building tree using sklearn. I would expect it to be much faster using vectorized code.

Part 8

a)

I use following code to get the first split data:

```
def random_train_top(x, y, maxweights):
    clf = tree.DecisionTreeClassifier(splitter='best', max_depth=1, /
    criterion='entropy', max_features=maxweights)
    clf = clf.fit(x, y)
    return clf
```

Then I used the same method the produce the tree structured graph:

```
import graphviz
from subprocess import call
clf = random_train_top(train_x, train_y, 100)
tree.export_graphviz(clf, out_file='tree0.dot', feature_names=words_order, max_depth=1,
                          class_names=['fake', 'real'])
call(['dot', '-Tpng', 'tree0.dot', '-o', 'tree0.png'])
```

Which produce a decision tree:

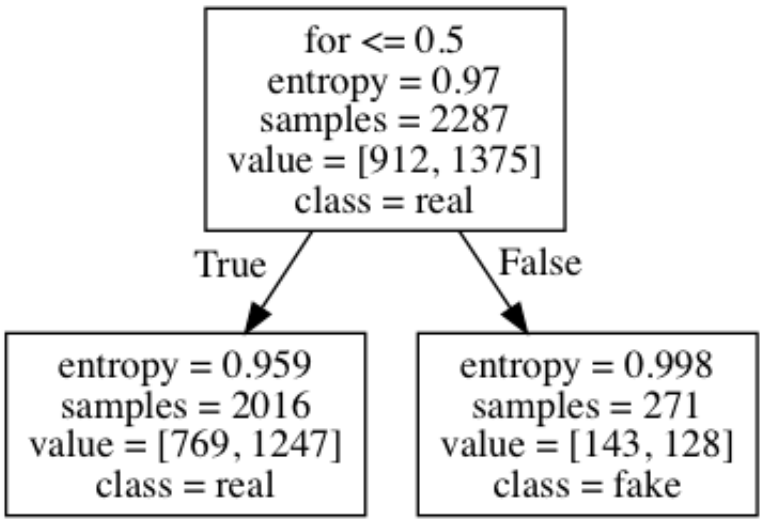

Figure 5: decision tree to compute I(Y,'for')

Here's how I compute:
Y=['fake','real']

H('for')=-(912/(912+1375))*$\log_2(912/(912+1375)) - (1375/(912+1375)) * log_2(1375/(912+1375)) = 0.9702$

$H('for'|Y) = (((769+1247)/(2287)) * (-(769/(769+1247)) * log2(769/(769+1247)) - (1247/(769+1247)) * log2(1247/(769+1247)))) + (((143+128)/(2287)) * (-(143/(143+128)) * log2(143/(143+128)) - (128/(143+128)) * log2(128/(143+128)))) = 0.9636$

$I(Y,'for') = 0.9702 - 0.9636 = 0.0066$

b)

Similarly, using the above code, I generated another decision tree with a different $x_j =' a'$:

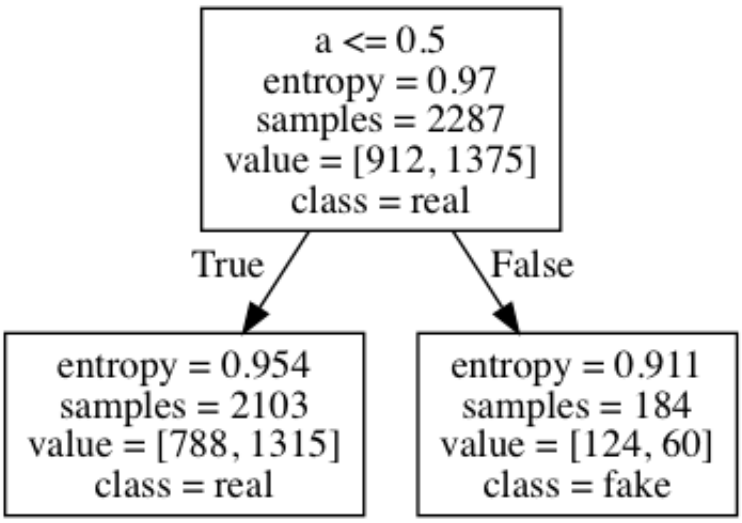

Figure 6: decision tree to compute I(Y,'for')

Here's how I compute:
Y=['fake','real']

H('a')=-(912/(912+1375))*$\log_2(912/(912+1375)) - (1375/(912+1375)) * log_2(1375/(912+1375)) = 0.9702$

$H('a'|Y) = (((788+1315)/(2287)) * (-(788/(788+1315)) * log2(788/(769+1315)) - (1315/(788+1315)) * log2(1315/(788+1315)))) + (((124+60)/(2287)) * (-(124/(124+60)) * log2(124/(124+60)) - (60/(124+60)) * log2(128/(124+60)))) = 0.9175$

$I(Y,' a') = 0.9702 - 0.9175 = 0.0527$

In conclusion, I(Y,'a') is bigger than I(Y,'for') which indicates we can learn more information by using 'a' as split thus its better