# CSC411: Project #3

Due on Monday, March 19, 2018

**Yu-Chien Chen, Hunter Richards**

Saturday, March 17, 2018

## Part 1

The aim of the project is to build and analyze several algorithms for determining if a headline is real or fake news. There are 1298 fake news headlines and 1968 real news headlines in the dataset and the headlines were cleaned up using clean_script.py. The two datasets were split randomly into 3 sets: 70% for training set, 15% each for validation and test set.

It is feasible to predict whether or not the headline is real or fake by using simple statistical methods. One can look at the difference between the probability of the word being in a real headline and the probability of it being in a fake headline and then using that to determine how much more likely the word is to be in real news rather than in fake news. For example, the three words that are strong indicators of real or fake news are: "donald", "the", and "trumps" since the difference between their probability of being in real or fake news is the largest of all words in the vocabulary. That is to say, they are the most decisive words to look at when trying to predict if a headline is real or fake.

| | Word | P(word **in** real news) | P(word **in** fake news) |
|---|---|---|---|
| 1. | 'donald' | 0.415 | 0.183 |
| 2. | 'the' | 0.074 | 0.273 |
| 3. | 'trumps' | 0.110 | 0.002 |

If the headline contains the word "donald", it is more likely that the news is real since "donald" appears more often in real news' headlines. On the other hand, "the" has a higher appearance rate in fake than real news.

The rest of this report is based upon a dataset that was randomly split into $\sim$ 70% training, $\sim$ 15% validation, and $\sim$ 15% test. All code used in this assignment is implemented in Python 3.6.

## Part 2

The Naive Bayes (NB) classifier classifies the headlines by comparing the probability of the news being fake given that the word $x$ shows up in the headline $P(y=1|x)$ and the probability that the news is real given that the same x shows up in the headline$P(y=0|x)$. If the probability of it being real is higher than it being fake, than NB classifies it as real news and vice versa. Bayes rules comes in when calculating $P(y|x)$. For example:

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)}$$

$$= \frac{(\prod_{i-1}^{n} P(x_i|y=1))P(y=1)}{(\prod_{i-1}^{n} P(x_i|y=1))P(y=1) + (\prod_{i-1}^{n} P(x_i|y=0))P(y=0)}$$

To model $P(x|y)$, naive bayes uses the assumption that $x_i$'s are conditionally independent given y. Therefore,

$$P(x_1, x_2, ..., x_n|y) = P(x_1|y)P(x_2|y, x_1)P(x_3|y, x_1, x_2)...P(x_n|y, x_1, ..., x_{n-1})$$

$$= P(x_1|y)P(x_2|y)P(x_3|y)...P(x_n|y)$$

$$= \prod_{i=1}^{n} P(x_i|y)$$

Since each $P(x_n|y)$ may be really small, multiplying them leads to underflow. Therefore, we utilized the fact that $a_1a_2...a_n = exp(\log a_1 + \log a_2 + ... + \log a_n)$ to avoid underflow.

The training set $X$ for this algorithm is an n by m matrix where n is the number for headlines and m is the number of words in the feature vector (aka the dictionary). The feature vector, $x$, contains all the words that appeared in the headlines. For example:

$$x = \begin{bmatrix} 1 & 0 & 0 & \dots & 1 & \dots & 0 \end{bmatrix}$$

where each column represent a word in the dictionary and if the headline contains the i-th word of the dictionary, we set $x_i = 1$; otherwise, we set $x_i = 0$.

The Naive Bayes algorithm on the training set gives 97.55% accuracy, and the final performance on the test set is 82.82% after tuning the parameters of the prior $m$ and $\hat{p}$. The final values for $m$ and $\hat{p}$ are 1 and 0.069, respectively. The two values are obtained through grid search to find the numbers that give the best performance.

## Part 3

a) Top 10 words whose presence most strongly predicts that the news is real were found by calculating $P(y = 0|x_i = 1)$ for all the words in the dictionary and select the top 10 words that has the highest probability.

$$P(y = 0|x_i = 1) = \frac{P(x_i = 1|y = 0)P(y = 0)}{P(x_i)}$$

The words turned out to be:

```
1.  'korea'         6.  'paris'
2.  'turnbull'      7.  'trumps'
3.  'travel'        8.  'under'
4.  'australia'     9.  'congress'
5.  'debate'        10. 'refugee'
```

(The rest were obtained in a similar fashion.)

Top 10 words whose absence most strongly predicts that the news is real were found by calculating $P(y = 0|x_i = 0)$ and the words are:

```
1.  'trump'         6.  'is'
2.  'the'           7.  'and'
3.  'to'            8.  'for'
4.  'a'             9.  'in'
5.  'hillary'       10. 'of'
```

Top 10 words whose presence most strongly predicts that the news is fake were found by calculating $P(y = 1|x_i = 1)$ and the words are:

```
1.  'breaking'      6.  'woman'
2.  'reporter'      7.  '3'
3.  'soros'         8.  'd'
4.  'u'             9.  'info'
5.  'liberty'       10. '4'
```

Top 10 words whose absence most strongly predicts that the news is fake were found by calculating $P(y = 1|x_i = 0)$ and the words are:

```
1.  'donald'        6.  'korea'
2.  'trumps'        7.  'ban'
3.  'us'            8.  'turnbull'
4.  'says'          9.  'travel'
5.  'north'         10. 'wall'
```

The top 10 words whose presence most strongly predict that the news is real are very similar to the top 10 whose absence most strongly predict fake news. The presence of a word has a stronger influence in predicting whether or not the news is real or fake since $P(c|x = 1)$ (generally around 0.98) is greater than $P(c|x = 0)$ (generally around 0.4 to 0.6).

b) Stop words like ”a” and ”to” were removed by importing:

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
```

Now the new top 10 words that most strongly predict that the news is real are:

1. 'korea'
2. 'turnbull'
3. 'travel'
4. 'australia'
5. 'debate'
6. 'paris'
7. 'trumps'
8. 'congress'
9. 'refugee'
10. 'ahead'

Top 10 non-stopwords whose absence most strongly predicts that the news is real are:

1. 'trump'
2. 'hillary'
3. 'just'
4. 'clinton'
5. 'new'
6. 'america'
7. 'watch'
8. 'obama'
9. 'win'
10. 'voter'

Top 10 non-stopwords that most strongly predict that the news is fake are:

1. 'breaking'
2. 'reporter'
3. 'soros'
4. 'u'
5. 'liberty'
6. 'woman'
7. '3'
8. 'd'
9. 'info'
10. '4'

Top 10 non-stopwords whose absence most strongly predicts that the news is fake are:

1. 'donald'
2. 'trumps'
3. 'says'
4. 'north'
5. 'korea'
6. 'ban'
7. 'turnbull'
8. 'travel'
9. 'wall'
10. 'australia'

c) It makes sense to remove the stop-words since they are the most common words used in sentences and they usually do not indicate any meaning when trying to decide if the news is real or fake. Since stop-words do not help in differentiating between real or fake news, it would be reasonable to remove them in order to get a more accurate result.

## Part 4

The logistic regression model was constructed using PyTorch. It consists of a dropout, linear, and sigmoid activation layer. The weights were initialized using Xavier initialization and the biases were initialized to a small constant value. Cross entropy loss was used to train the model. Several methods of regularization were considered including: L1, L2, early stopping, and dropout. Before methods can be compared, a baseline was established. With no regularization, the dropout layer removed, and training until 100% training set accuracy, the following learning curve was produced:

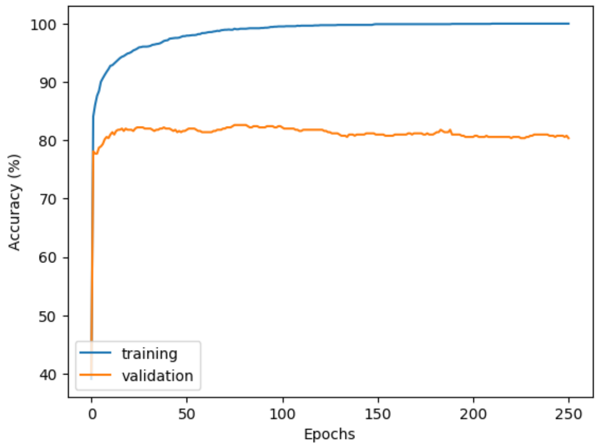

Figure 1: Logistic regression learning curve, 250 epochs, no regularization

There is a slight decrease in validation set accuracy indicating overfitting. Each of L1 and L2 regularization were tested to see which one produced the best results. Starting with a large value of $\lambda = 0.1$ and then decrementing in order of magnitude, a range of values were tested. L2 regularization gave the best results on the validation set. PyTorch includes the built-in L2 regularization parameter `weight_decay` as part of the Adam optimizer. Both `weight_decay` and the manual implementation of L2 regularization performed identically, so the easy-to-implement PyTorch parameter was chosen. Using the experimentally obtained value $10^{-3}$, the following learning curve was produced:

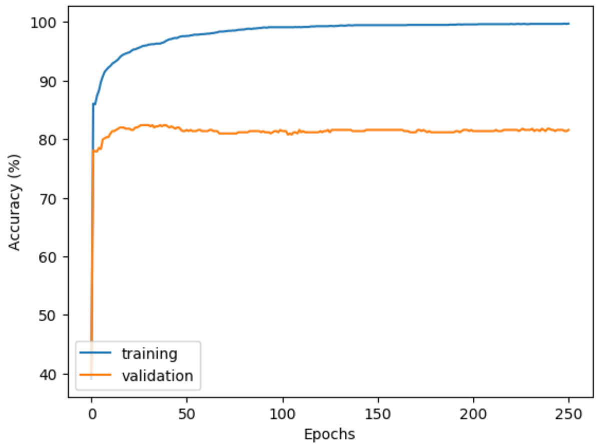

Figure 2: Logistic regression learning curve, 250 epochs, with L2

The decrease in accuracy on the validation set is now virtually absent. Adding a dropout layer before the linear layer in the model improved this curve further:

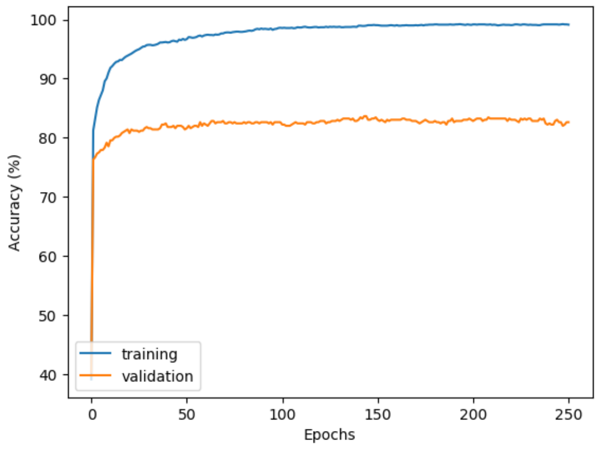

Figure 3: Logistic regression learning curve, 250 epochs, with L2 and dropout

There is, however, more fluctuation in the curve which is to be expected with dropout. Early stopping was employed to further increase the performance of the model. The model was stopped earlier and earlier until performance decreased, and then the previous stopping point was chosen. This process was repeated several times to ensure the fluctuations were not causing the decrease in performance. The optimal stopping point was approximately at the same location ($80 \pm 10$ epochs) each time. This produced the final learning curve:

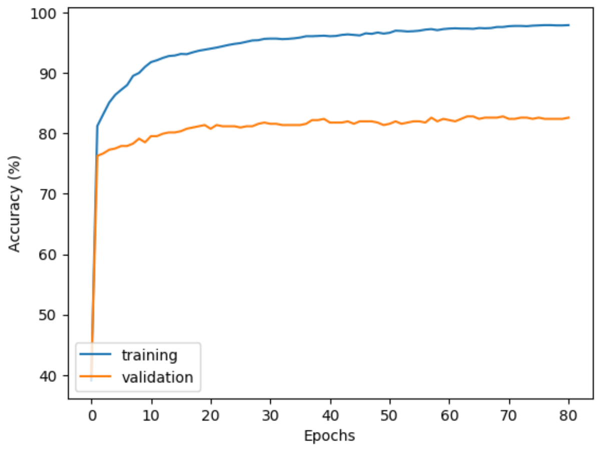

Figure 4: Logistic regression learning curve, 80 epochs, with L2 and dropout

The accuracy of the model was 97.94% on the training set and 84.66% on the test set.

## Part 5

In the Naive Bayes model, the news is predicted to be real when $P(y=0|x_1,...,x_n) > P(y=1|x_1,...,x_n)$.

$$\frac{P(y=0|x_1,...,x_n)}{P(y=1|x_1,...,x_n)} > 1$$

$$\log\frac{P(y=0|x_1,...,x_n)}{P(y=1|x_1,...x_n)} > 0$$

$$\log\frac{P(y=0)\prod_j P(x_i|y=0)}{P(y=1)\prod_j P(x_i|y=1)} > 0$$

$$\log\frac{P(y=0)}{P(y=1)} + \sum_j \log\frac{P(x_j|y=0)}{P(x_j|y=1)} > 0$$

$$\beta_0 + \sum_j \beta_j x_j > 0$$

where

$$\beta_0 = \log\frac{P(y=0)}{P(y=1)} + \sum_j \log\frac{P(x_j=0|y=0)}{P(x_j=0|y=1)}$$

is the bias term, and

$$\beta_j = \log\frac{P(x_j=1|y=0)}{P(x_j=1|y=1)} - \log\frac{P(x_j=0|y=0)}{P(x_j=0|y=1)}$$

can be thought of as comparing the relative likelihoods of that feature being present/absent, conditioned on the two classes. In this case, $\theta_i = \beta_j$ and $I_i(x) = x_j$ and the threshold thr $= 0$.

The logistic regression model is typically formulated as:

$$z = \boldsymbol{\theta}^T \mathbf{x} + \theta_0$$

$$y = \sigma(z)$$

where the output $y$ is compared to some threshold $r$.

This can be re-written to match the given form:

$$\begin{aligned} y &> r \\ \sigma(z) &> r \\ \frac{1}{1+e^{-z}} &> r \\ -\log\left(\frac{1}{r}-1\right) &> z \\ z &> \log\left(\frac{1}{r}-1\right) \\ \boldsymbol{\theta}^T \mathbf{x} + \theta_0 &> \text{thr} \\ \theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k &> \text{thr} \\ \theta_0 + \theta_1 \mathbf{1}_1{}^T \mathbf{x} + \cdots + \theta_k \mathbf{1}_k{}^T \mathbf{x} &> \text{thr} \\ \theta_0 + \theta_1 I_1(\mathbf{x}) + \cdots + \theta_k I_k(\mathbf{x}) &> \text{thr} \end{aligned}$$

where $\mathbf{1}_i$ is a one-hot encoded column vector (the $i$-th entry is a one) and $\text{thr} = \log\left(\frac{1}{r}-1\right)$.

From this we see that $\theta_0$ is the bias term of the network, $\theta_1 \cdots \theta_k$ are the weights, and $I_1(\mathbf{x}) \cdots I_k(\mathbf{x})$ are functions of the input vector $\mathbf{x}$ whereby the input is multiplied by a one-hot vector to extract a single feature. Each extracted feature is either 0 or 1 and is multiplied by a weight $\theta_i$, which is its relative importance to determining the input's classification.

## Part 6

a) The top ten positive $\theta$'s and their corresponding words obtained from the logistic regression model were:

```
1.  (1.1639862, 'breaking')      6.  (0.9928428 , 'u')
2.  (1.1056182, 'watch')         7.  (0.980342  , '3')
3.  (1.0237961, 'soros')         8.  (0.9201495 , 'this')
4.  (1.0173825, 'reporter')      9.  (0.91574365, '4')
5.  (0.9954617, 'black')         10. (0.8882559 , 'voter')
```

The top ten negative $\theta$'s and their corresponding words were:

```
1.  (-1.5508783, 'trumps')       6.  (-1.1072944 , 'travel')
2.  (-1.3265325, 'korea')        7.  (-0.97115254, 'business')
3.  (-1.3209461, 'turnbull')     8.  (-0.97054833, 'congress')
4.  (-1.1452451, 'australia')    9.  (-0.96658   , 'ban')
5.  (-1.111501 , 'debate')       10. (-0.96304804, 'paris')
```

The top ten positive $\theta$'s strongly resemble the top ten words whose presence indicate the news is fake from Part 3(a). 8 out of the 10 words in either list match. Likewise, the top ten negative $\theta$'s resemble the top ten words whose presence indicate the news is real. However, only 6 out of 10 words match, yet still a considerable amount. This makes sense considering an output of 1 is classified as "fake" and an output of 0 is classified as "real". Negative weights will drive the output towards 0 when their corresponding words are present, and vice versa. The ordering of the words are different for both models. This indicates that the models weigh the words differently in order to classify an input, and may prioritize some words that the other model does not.

b) The top ten positive $\theta$'s and their corresponding words obtained from the logistic regression model were:

```
1.  (1.1639862, 'breaking')      6.  (0.9928428 , 'u')
2.  (1.1056182, 'watch')         7.  (0.980342  , '3')
3.  (1.0237961, 'soros')         8.  (0.91574365, '4')
4.  (1.0173825, 'reporter')      9.  (0.8882559 , 'voter')
5.  (0.9954617, 'black')         10. (0.872209  , 'just')
```

The top ten negative $\theta$'s and their corresponding words were:

```
1.  (-1.5508783, 'trumps')       6.  (-1.1072944 , 'travel')
2.  (-1.3265325, 'korea')        7.  (-0.97115254, 'business')
3.  (-1.3209461, 'turnbull')     8.  (-0.97054833, 'congress')
4.  (-1.1452451, 'australia')    9.  (-0.96658   , 'ban')
5.  (-1.111501 , 'debate')       10. (-0.96304804, 'paris')
```

The top ten positive $\theta$'s remain largely unchanged, while the top ten negative $\theta$'s did not contain any stop-words to begin with and are therefore, unchanged. This change did not introduce any more words that are present in both the lists from Part 3(b) and the lists here. Most of what can be said of the similarities/differences between Part 3(b)'s lists and these lists has already been mentioned in Part 6(a).

c) Using the magnitude of the logistic regression parameters to indicate importance of a feature is a bad idea in general because difference features may not be of the same scale. The cost function will be greatly asymmetric and the weights of these extreme-valued features cannot be used to accurately assess the features' importance. The features must be normalized so that they are all scaled equally, and therefore their weights are scaled equally.

   In this project, the features have binary values and are already in a sense "normalized". So, it is okay to look at the raw weights to determine importance without explicitly performing feature normalization.

## Part 7

a) Recording the performance of the decision tree on the training and validation sets for a range of depth values produced the following graph:

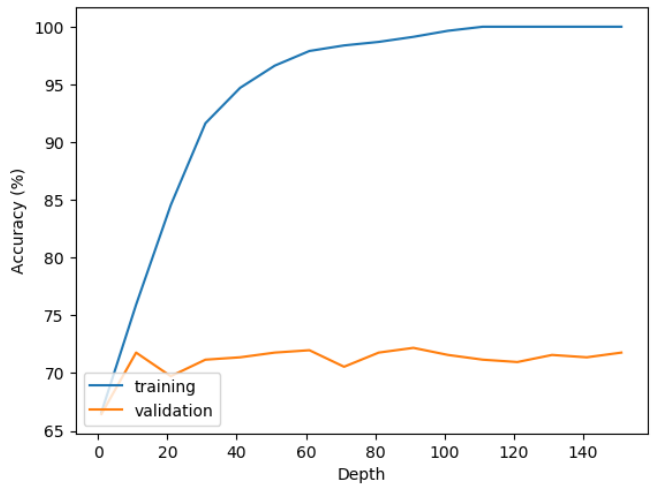

Figure 5: Relationship between depth and accuracy of the decision tree classifier

Depth for decision trees appears to serve the same role and number of epochs for training neural networks: increasing the depth results in a generally higher training and validation set accuracy until a point where the model starts to overfit. Also, the validation set accuracy is always lower than the training set accuracy which is typical of the machine learning models covered so far in the course.

From testing the range of values, a maximum depth of 151 gave the best validation set results. The other parameter that was changed from its default value was `max_features`. It caused the most significant increase in performance and was selected through similar fashion. The optimal value determined was `max_features = 310`. Adjusting the maximum number of features used in determining the split at each level was likely preventing the model from overfitting. Less features to look at means the model is less capable of memorizing the training set (i.e. the model's capacity has been reduced to improve generalization). The resulting decision tree returned an accuracy of 100.00%, 75.26%, and 77.71% on the training, validation, and test set respectively.

b) The following visualization was extracted from the decision tree from Part 7(a):

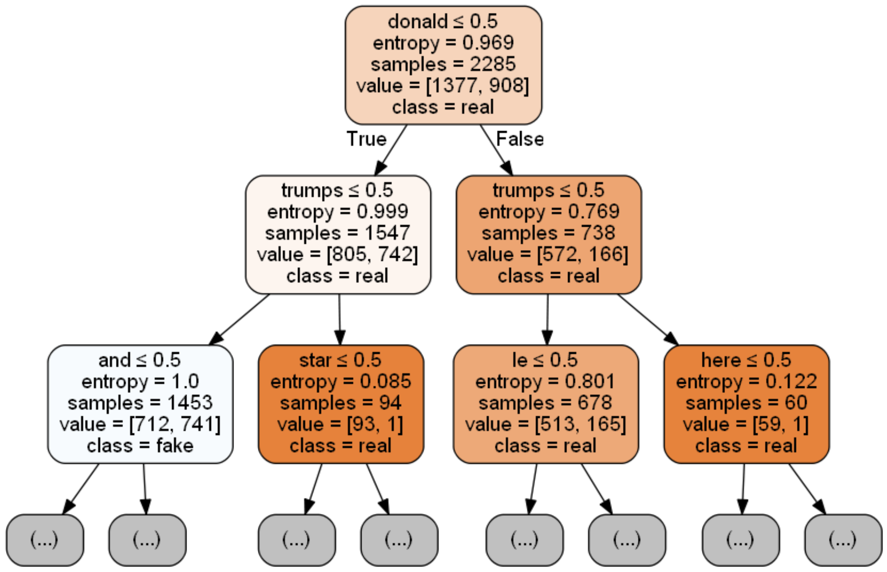

Figure 6: Decision tree classifier, first two levels

Most of the words deemed important by the decision tree (i.e. the words present in the highest levels) do not appear in the other models, save `'trumps'` which has shown up in every model discussed and `'donald'` which was the number one word whose absence most strongly predicted the news was fake for the Naive Bayes model. The word `'trumps'` has been used as a top word for determining if the news is real in both Part 3(a) and 6(a), and even in Part 1.

It appears as though some words are especially important for classifying a headline and always show up, but overall the decision tree uses very different methods for classifying the headlines based on the words it chose.

c) The performance of the three models tested is summarized in the following table:

| Classifier | Training set acc. | Validation set acc. | Test set acc. | Time to train |
|---|---|---|---|---|
| Naive Bayes | 0.9755 | 0.8200 | 0.8282 | 58.44 s |
| Logistic Regression | 0.9794 | 0.8262 | 0.8466 | 51.28 s |
| Decision Tree | 1.0000 | 0.7526 | 0.7771 | 0.73 s |

The Logistic Regression classifier performed the best (only slightly better than Naive Bayes) while the Decision Tree classifier performed the worst. However, this is only considering raw accuracy on the test set. It should be noted that the decision tree classifier was exceptionally fast to train, approximately 70 times faster than Logistic Regression and Naive Bayes. This may be desirable when accuracy can be traded for faster compute times.

It appears that the Decision Tree classifier overfit the most because it has the largest discrepancy between its training set accuracy and its validation set accuracy. The training set accuracy is perfect (which could be due to the model memorizing the training set), while the validation set accuracy is a lackluster 75.26%. Perhaps with a better understanding of `skylearn`'s decision tree, or through manual implementation, the model could be made to perform better.

## Part 8

Mutual information was obtained by calculating

$$I(Y, x_i) = H(Y) - H(Y|x_i)$$

where $Y$ is the random variable that indicates whether or not the news is real or fake, and $x_i$ is the word chosen for the split. The code used to calculate $I(Y, x_i)$ is shown below:

```
def I(vocab, word, x):
    real_x1 = 0
    real_x0 = 0
    fake_x1 = 0
    fake_x0 = 0
    total = float(len(x))
    midpoint = int(NUM_REAL*SET_RATIO[0])
    vocab = list(vocab.keys())
    index = vocab.index(word)
    for i, n in enumerate(x):
        if i < midpoint:
            if n[index] == 1:
                real_x1 += 1
            else:
                real_x0 += 1
        else:
            if n[index] == 1:
                fake_x1 += 1
            else:
                fake_x0 += 1
    h_Y = H(real_x1 + real_x0, fake_x1 + fake_x0)
    h_YXi = (real_x1 + fake_x1)/total * H(real_x1, fake_x1) +
            (real_x0+fake_x0)/total * H(real_x0, fake_x0)
    mutualInfo = h_Y - h_YXi
    print(mutualInfo)
    return (mutualInfo)

def H(x1, x2):
    total = float(x1 + x2)
    return -x1/total * np.log2(x1/total)-x2/total * np.log2(x2/total)
```

The mutual information between Y and the word ”donald” was calculated to be 0.0447.
Another word, ”star”, that is different from the first split word was chosen for a comparison. Using the same code, $I(Y, star)$ was calculated to be 0.0085.

As expected, any other words that is not the first split word would have a smaller mutual information value than $I(Y, donald)$ since the decision tree's `splitter` parameter was set to `best`. This means the tree will choose the split word that gives the highest information gain (i.e. the most decisive feature) at each tree node. The feature chosen for the root of the tree will therefore be the most decisive word in the entire vocabulary.