CSC411: Assignment #3

Due on Monday, March 19, 2018

Yifei Dong, Zifei Han

March 19, 2018

Part 1

Dataset description
The raw data are lines of tittles. Each line is a different tittle. We stores each line as a list stacked in an numpy arrays called real and fake. We then separate the numpy arrays into training, validation and test sets.

Since we want to predict whether a headline is fake or real by the words in it. We wanted to know that if different words have different number of occurrences in fake and real datasets. The number of occurrences of a word is the number of headlines that includes the word. Below are the number of occurrences we obtained.

From the data obtained, we can see that some words may be used to predict whether a headline is real or fake. For example, the number of occurrences of words "hillary", "trumps" and "says" are really different in real and fake datasets.

The number of occurrences of the words in the **real** headlines are:

- ('hillary', 24)
- ('trumps', 219)
- ('says', 178)

The number of occurrences of the words in the **fake** headlines are:

- ('hillary', 150)
- ('trumps', 4)
- ('says', 46)

Part 2

Implement the Naive Bayes algorithm for predicting whether a headline is real or fake

To tune the parameters of the prior, we tried several different values for m and p. We found out that m = 1 and p = 0.1 gave the best for performance for validation set. For example, validation set has performance 76% for m = 2 and p = 1, which is much lower than then performance we obtained with m=1 and p=0.1. And when we set m = 0.5 and p = 0.01, the performance of the validation set is 85%.

Below are the performances of all sets when m=1 and p=0.1:
The performance of the Naive Bayes classifier on the training set is 97.19912472647702%
The performance of the Naive Bayes classifier on the validation set is 86.88524590163934%
The performance of the Naive Bayes classifier on the test set is 86.09406952965234%

In Naive Bayes classifier, one of the steps is to compute $P(word1, word2, word3, ..., wordn|class)$, where wordi are all the words, and class is either fake or real. Since conditional independence is assumed under Naive Bayes algorithm, instead of computing $P(word1, word2, word3, ..., wordn|class)$, we are commputing $\prod_{i=1}^{n} P(wordi|class)$. $P(wordi \mid class)$ is the probability that the word appears when class is either fake or real. It is a very small number between 0 and 1. Computing products of many small numbers leads to underflow. Therefore, we used the fact that $a_1 a_2 ... a_n = exp(loga_1 + loga_2 + ... + loga_n)$.

```
multi_real = 0
for p in prob_word_real:
    multi_real += math.log(p)
multi_real = math.exp(multi_real)
```

This is how we used the fact in the code. prob_word_real is a list containing the probabilities of $P(wordi|real)$ for all words. We set multi_real = 0 first. And then add all $log(P(wordi|real))$ together by a for loop. The result is then computed by math.exp().

Part 3

Part A

Each word is listed with its probability.
List the 10 words and $P(real|word)$ whose presence most strongly predicts that the news is real,:
('q', 0.9998993370668853)
('x', 0.9997159529917837)
('ww', 0.9997159529917835)
('w', 0.9997159529917835)
('hrw', 0.9997159529917835)
('wh', 0.9997159529917835)
('why', 0.9997159529917835)
('nsw', 0.9994293650216491)
('qna', 0.9992099727162758)
('asx', 0.9988907972131266)

List the 10 words and $P(real| \sim word)$ whose absence most strongly predicts that the news is real:
('laugh', 0.6026258206279007)
('rural', 0.6026258206279007)
('usa', 0.6026258206279007)
('usas', 0.6026258206279007)
('saul', 0.6026258206279007)
('maga', 0.6026258205947712)
('ham', 0.6026258205947712)
('graham', 0.6026258205947712)
('llama', 0.6026258205947712)
('alarm', 0.6026258205947712)

List the 10 words and $P(fake|word)$ whose presence most strongly predicts that the news is fake:
('communicated', 0.9999999999931778)
('unconfirmed', 0.9999999999680339)
('boycottcomedian', 0.9999999998984695)
('unpredictability', 0.9999999998790827)
('unpredictable', 0.9999999998790827)
('manipulated', 0.9999999997823266)
('productive', 0.9999999996724875)
('motherfucker', 0.9999999994414386)
('trumprolled', 0.9999999992465408)
('misconduct', 0.9999999991492835)

List the 10 words and $P(fake| \sim word)$ whose absence most strongly predicts that the news is fake:
('00', 0.39738409697665245)
('ll', 0.39738409697665245)
('000', 0.39738409697665245)
('yr', 0.39738409697665245)
('ssy', 0.39738409697665245)
('harry', 0.3973745732738635)
('ass', 0.3973745732738635)
('harass', 0.3973745732738635)

('gay', 0.3973745732738635)
('salsa', 0.3973745732738635)

We changed our code in part2 a bit in order to get the probabilities $P(class|word)$ and $P(class| \sim word)$ for each word from Naive Bayes classifier, where class is either real or fake. Conditional probabilities $P(word|fake)$ and $P(word|real)$ are used here

We computed $P(fake|word)$ by computing $P(word|fake) \times P(fake)$ and $P(word|real) \times P(real)$ first.
Then $P(fake|word) = \dfrac{P(word|fake) \times P(fake)}{P(word|fake) \times P(fake) + P(word|real) \times P(real)}$
$P(real|word) = 1 - P(fake|word)$

And $P(fake| \sim word)$ is computed by:
$P(fake| \sim word) = \dfrac{(1 - P(word|fake)) \times P(fake)}{(1 - P(word|fake)) \times P(fake) + (1 - P(word|real)) \times P(real)}$
$P(real| \sim word) = 1 - P(fake| \sim word)$

From the probabilities, we can see that the $P(class|word)$ are close to 1 for the words listed above and $P(class| \sim word)$ are around 0.6 and 0.3. Therefore, the influence of presence on predicting whether the headline is real or fake is much bigger.

Part B

After get rid of the stop words:
List the 10 words whose presence most strongly predicts that the news is real:
('q', 0.9998993370668853)
('x', 0.9997159529917837)
('ww', 0.9997159529917835)
('w', 0.9997159529917835)
('hrw', 0.9997159529917835)
('wh', 0.9997159529917835)
('nsw', 0.9994293650216491)
('qna', 0.9992099727162758)
('asx', 0.9988907972131266)
('hallways', 0.9988907972131257)

List the 10 words whose absence most strongly predicts that the news is real:
('laugh', 0.6026258206279007)
('rural', 0.6026258206279007)
('usa', 0.6026258206279007)
('usas', 0.6026258206279007)
('saul', 0.6026258206279007)
('maga', 0.6026258205947712)
('ham', 0.6026258205947712)
('graham', 0.6026258205947712)
('llama', 0.6026258205947712)
('alarm', 0.6026258205947712)

List the 10 words whose presence most strongly predicts that the news is fake:
('communicated', 0.9999999999931778)

('unconfirmed', 0.9999999999680339)
('boycottcomedian', 0.9999999998984695)
('unpredictability', 0.9999999998790827)
('unpredictable', 0.9999999998790827)
('manipulated', 0.9999999997823266)
('productive', 0.9999999996724875)
('motherfucker', 0.9999999994414386)
('trumprolled', 0.9999999992465408)
('misconduct', 0.9999999991492835)

List the 10 words whose absence most strongly predicts that the news is fake:
('00', 0.39738409697665245)
('ll', 0.39738409697665245)
('000', 0.39738409697665245)
('yr', 0.39738409697665245)
('ssy', 0.39738409697665245)
('harry', 0.3973745732738635)
('ass', 0.3973745732738635)
('harass', 0.3973745732738635)
('gay', 0.3973745732738635)
('salsa', 0.3973745732738635)

Part C

Why might it make sense to remove stop words when interpreting the model?
The stop words does not have an impact on the meaning of the title.

Why might it make sense to keep stop words?
The number of occurrences of stop words might be different in real and fake headlines. So they can still be used to predict whether the headline is fake or real.

Part 4

The logistic regression training algorithm is implemented with pytorch.
1. The parameters are selected as follows:

- *init_t* - initialized weights using all zeros of vector.
- α - tried different alphas including 0.00001, 0.001, 0.0001
  selected 0.001 because it results the best result
- loss function
  CrossEntropyLoss is the correct loss function for logistic regression
- optimizer
  Used torch.optim.Adam optimizer. Tried several optimizers with l1, l2, and weight decay regularization none of them showed better result. So decided not to use regularization.

```
class LogisticRegression(torch.nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(input_size, num_classes)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(numIterations):
    y_pred = model(x_data)

    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

2. Plot learning curves

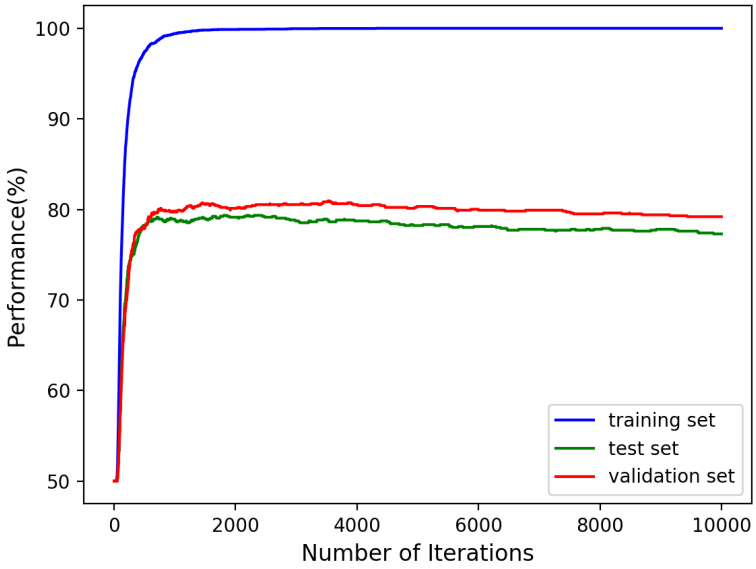


Figure 1

There are some overfiting in the result. This might because there are some words that dont overlap between training set and testing set. So its hard to be unbiased

Part 5

1. $I(x)$
$I(x)$ is the transformation of x. Both algorithm uses x as a word indicator, so each $I(x)$ represents a word that could appear in each deadline.
$I(x)$ indicates if the word exists in the headline for both algorithms."1" means exists and "0" means does not exist.

2. Theta
Theta is the coefficient associated with each keywords. It indicated how strong each word is associated with real news.

In the logistic regression:
$\theta_0 = -\beta_0$; $\theta_j = -\beta_j$
where β are the weights obtained in logistic regression.

In Naive Bayes, let label=1 indicate real class, label=0 indicate fake class:
$$\theta_0 = log\frac{P(label = 1)}{P(label = 0)} + \sum_j log\frac{P(\sim word_j|label = 1)}{P(\sim word_j|label = 0)}$$
$$\theta_j = log\frac{P(word_j|label = 1)}{P(word_j|label = 0)} - log\frac{P(\sim word_j|label = 1)}{P(\sim word_j|label = 0)}$$

Part 6

part a

1. Display a list of top 10 positive θs and negative θs obtained from Logistic Regression.

```
List of Top 10 positive thetas:
[('naranja', 3.790480136871338),
 ('isolationist', 3.8191072940826416),
 ('receives', 3.915940284729004),
 ('mehr', 4.15025520324707),
 ('fiscal', 4.304937362670898),
 ('discussions', 4.411450386047363),
 ('sat', 4.597943305969238),
 ('israels', 5.127058029174805),
 ('blindsides', 5.830095291137695),
 ('politicians', 5.868911266326904)]
```

```
List of Top 10 negative thetas:
[('also', -5.424506664276123),
 ('slams', -4.225096702575684),
 ('certifies', -4.214896202087402),
 ('ire', -4.14245080947876),
 ('abetz', -4.086751937866211),
 ('lays', -4.081603050231934),
 ('manipulated', -4.012431621551514),
 ('belt', -3.978590488433838),
 ('quiff', -3.9434258937835693),
 ('zieht', -3.7901220321655273)]
```

The thetas in logistic regression indicate how strong this word relates to fake or real news. It is determined by frequency. For example, 'politicians' with the most positive theta has occurred at fake twice, but did not occur in real headlines.

If we read the words one by one, the words in part6(a) are different from the words in part3(a). However, there are some similarities. Most of the words in these two parts has a common characteristic: their number of occurrences in the training dataset is 1 or close to 1. And they most likely only appeared in one of the fake or real dataset.

part b

1. Filter our all the stop words, and Display a list of top 10 positive θs and negative θs

```
List of Top 10 positive thetas without STOP WORDS:
[('mistakes', 3.816910982131958),
 ('popular', 3.8433315753936768),
 ('barge', 3.926193952560425),
 ('horcruxes', 4.10868501663208),
 ('yearns', 4.167271137237549),
 ('kept', 4.412245273590088),
 ('merit', 4.586661338806152),
```

```
 ('business', 4.700799942016602),
 ('deplaned', 5.750209331512451),
 ('tactic', 5.823776721954346)]
```

```
List of Top 10 negative thetas without STOP WORDS:
[('knees', -5.50216007232666),
 ('checks', -4.245494842529297),
 ('cutting', -4.212165355682373),
 ('deby', -4.178613662719727),
 ('trumpflation', -4.178532123565674),
 ('petraeus', -4.094372272491455),
 ('actor', -4.0814056396484375),
 ('indonesian', -3.9252517223358154),
 ('governor', -3.9013733863830566),
 ('named', -3.75889253616333)]
```

The thetas in logistic regression indicate how strong this word relates to fake or real news. It is determined by frequency of occurrence in both real and fake. For example, 'knees' with the most negative theta has occurred at fake once, but did not occur in real headlines.
The words listed in part6(b) and part3(b) are different. However, there are some similarities. Most of the words in these two parts has a common characteristic: their number of occurrences in the training dataset is 1 or close to 1. And they most likely only appeared in one of the fake or real dataset.

part c

Using magnitude in the real world is impossible because we are unable to generate parameter magnitude of all the words. Even if we can we will run forever. It is reasonable to use in this problem because we are able to generate the normalized magnitude input with all scope of the train, validation, and testing set.

Part 7

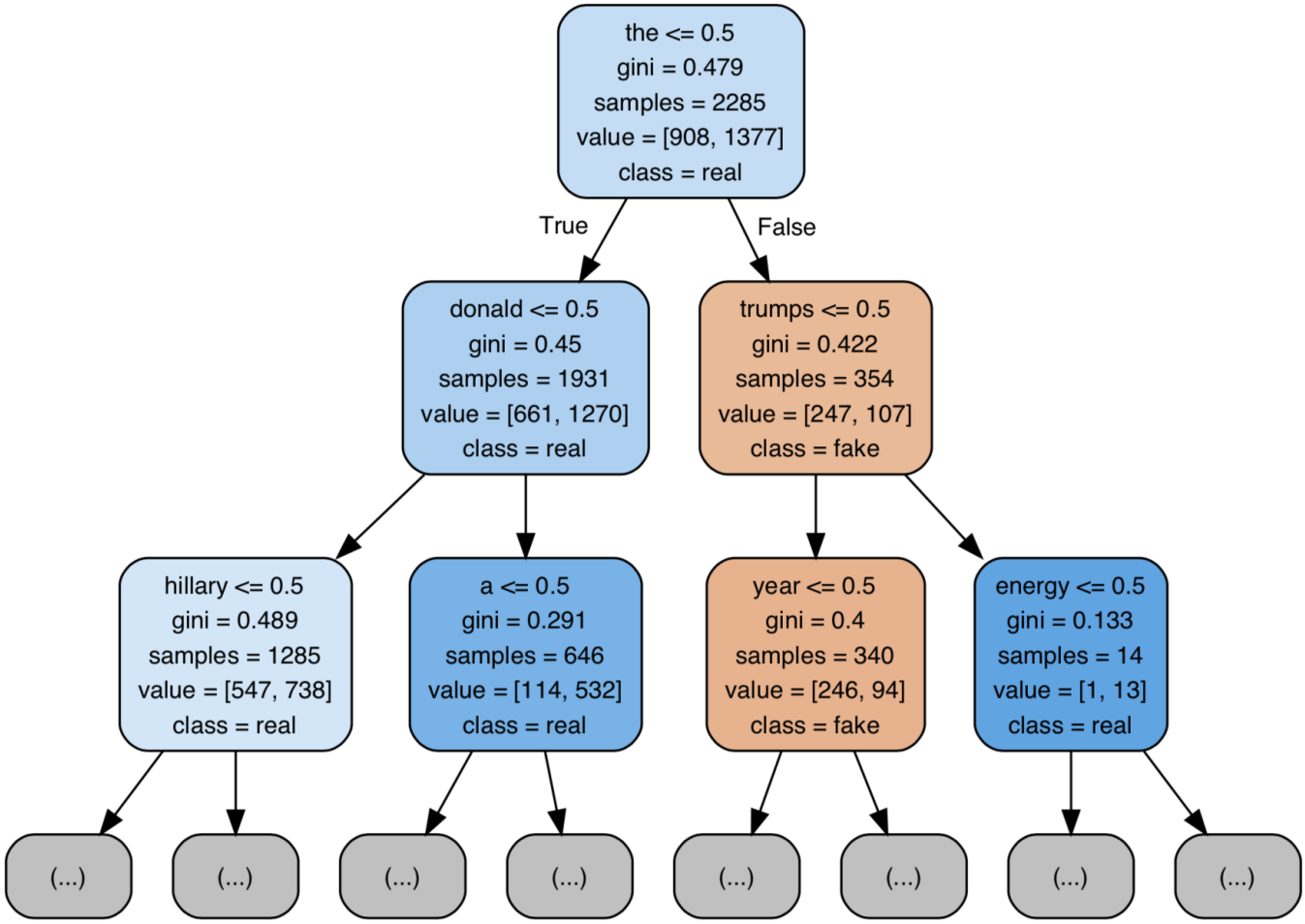
Part A

In order to choose a good paramter for max_depth. We tried different max_depth several times and picked the one which gives the highest performance for training and validation set.

```
Max depth is: 3
Training: 70.24070021881839 Validation: 69.4672131147541
Max depth is: 10
Training: 76.93654266958424 Validation: 72.1311475409836
Max depth is: 20
Training: 85.95185995623632 Validation: 74.38524590163934
Max depth is: 50
Training: 94.52954048140045 Validation: 74.59016393442623
Max depth is: 100
Training: 99.51859956236324 Validation: 75.81967213114754
Max depth is: 150
Training: 100.0 Validation: 76.84426229508196
Max depth is: 200
Training: 100.0 Validation: 77.04918032786885
Max depth is: 300
Training: 100.0 Validation: 76.4344262295082
Max depth is: 500
Training: 100.0 Validation: 76.84426229508196
Max depth is: 700
Training: 100.0 Validation: 76.4344262295082
```

The above is the relationship between the max_depth of the tree, and the training/validation accuracy. We can see that the decision tree has the highest accuracy when max depth is 200. For other parameters of the decision tree, we have tried change the min_samples_leaf, max_features and max_leaf_nodes but they didn't really improve the performance, so we just used the default ones.

Part B



Part C

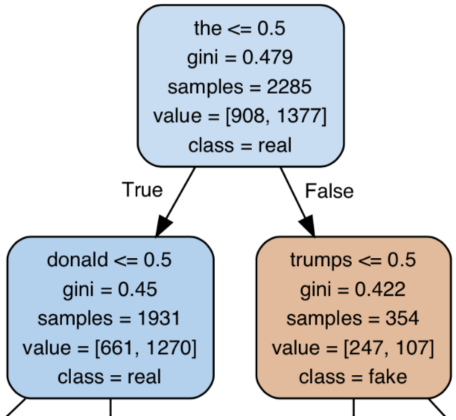
To Summarize:
On training set, the performance for Naive Bayes is 97%, the performance for Logistic Regression is nearly perfect for all iterations and the performance for Decision Tree is improving drastically with greater max_depth from 70% to 100%.

On validation set and testing set, the performances for Naive Bayes are both 86%, the performances for Logistic Regression are always around 80%, while for Decision Tree the performance for validation set increases with max_depth but converges at around 75%.

The best performance is Naive Bayes. The worst performance is Decision Tree. The algorithm that overfits the most is Logistic Regression.

Part 8

Part A



Want to compute $I(Y, x_i)$ for the first split of the decision tree. Therefore, x_i is the keyword "the". Y is either real or fake.

First compute the entropy:

$$\begin{aligned}H(Y) &= \sum_v -P(Y=v)log_2 P(Y=v)\\ &= -P(Y=fake)log_2 P(Y=fake) - P(Y=real)log_2 P(Y=real)\\ &= -\tfrac{908}{2285}log_2\tfrac{908}{2285} - \tfrac{1377}{2285}log_2 P(\tfrac{1377}{2285})\\ &\approx 0.529 + 0.4403\\ &= 0.9693\end{aligned}$$

$$\begin{aligned}H(Y|x_i = \neg the) &= -\tfrac{661}{1931}log_2\tfrac{661}{1931} - \tfrac{1270}{1931}log_2 P(\tfrac{1270}{1931})\\ &\approx 0.9270\end{aligned}$$

$$\begin{aligned}H(Y|x_i = the) &= -\tfrac{247}{354}log_2\tfrac{247}{354} - \tfrac{107}{354}log_2 P(\tfrac{107}{354})\\ &\approx 0.8840\end{aligned}$$

Also,
$P(x_i = \neg the) = 1931/2285 \approx 0.8451$
$P(x_i = the) = 354/2285 \approx 0.1549$

Then, the mutual information is:

$$\begin{aligned}I(Y, x_i) &= H(Y) - \sum_x P(x_i = x)H(Y|x_i = x)\\ &= H(Y) - [P(x_i = the)H(Y|x_i = the) + P(x_i = \neg the)H(Y|x_i = \neg the)]\\ &= 0.9693 - (0.1549 \times 0.8840 + 0.8451 \times 0.9270)\\ &= 0.0489607\end{aligned}$$

Part B

Below is the code we used to obtain the information required to compute $I(Y, x_j)$, where the keyword is 'trumps':

```
def part8b():
    real_with_word = []
    for headline in train_real:
        if "trumps" in list(headline):
            real_with_word.append(headline)

    with_word_real_split = len(real_with_word)
    without_word_real_split = len(train_real) - with_word_real_split

    fake_with_word = []
    for headline in train_fake:
        if "trumps" in list(headline):
            fake_with_word.append(headline)

    with_word_fake_split = len(fake_with_word)
    without_word_fake_split = len(train_fake) - with_word_fake_split

    print("Number of total headlines: " + str(len(train_real) + len(train_fake)))
    print("Number of initial fake headlines: " + str(len(train_fake)))
    print("Number of initial real headlines: " + str(len(train_real)))
    print("\n")
    print("Number of headlines without word 'trumps': " +
    str(without_word_fake_split + without_word_real_split))
    print("Number of fake headlines in the dataset without word 'trumps': " +
    str(without_word_fake_split))
    print("Number of real headlines in the dataset without word 'trumps': " +
    str(without_word_real_split))
    print("\n")
    print("Number of headlines with word 'trumps': " + str(with_word_fake_split +
    with_word_real_split))
    print("Number of fake headlines in the dataset with word 'trumps': " +
    str(with_word_fake_split))
    print("Number of real headlines in the dataset with word 'trumps': " +
    str(with_word_real_split))
```

The output is:
Number of total headlines: 2285
Number of initial fake headlines: 908
Number of initial real headlines: 1377

Number of headlines without word 'trumps': 2132
Number of fake headlines in the dataset without word 'trumps': 905
Number of real headlines in the dataset without word 'trumps': 1227

Number of headlines with word 'trumps': 153
Number of fake headlines in the dataset with word 'trumps': 3
Number of real headlines in the dataset with word 'trumps': 150

Use the same method to compute $I(Y, x_j)$ as how we computed $I(Y, x_i)$ in part A:

First compute the entropy:

$$\begin{aligned}
H(Y) &= \sum_v -P(Y=v) log_2 P(Y=v) \\
&= -P(Y=fake) log_2 P(Y=fake) - P(Y=real) log_2 P(Y=real) \\
&= -\tfrac{908}{2285} log_2 \tfrac{908}{2285} - \tfrac{1377}{2285} log_2 P(\tfrac{1377}{2285}) \\
&\approx 0.529 + 0.4403 \\
&= 0.9693
\end{aligned}$$

$$\begin{aligned}
H(Y|x_j = \neg trumps) &= -\tfrac{905}{2132} log_2 \tfrac{905}{2132} - \tfrac{1227}{2132} log_2 P(\tfrac{1227}{2132}) \\
&\approx 0.9835
\end{aligned}$$

$$\begin{aligned}
H(Y|x_j = trumps) &= -\tfrac{3}{153} log_2 \tfrac{3}{153} - \tfrac{150}{153} log_2 P(\tfrac{150}{153}) \\
&\approx 0.1392
\end{aligned}$$

Also,
$P(x_j = \neg trumps) = 2132/2285 \approx 0.9331$
$P(x_j = trumps) = 153/2285 \approx 0.0669$

Then, the mutual information is:

$$\begin{aligned}
I(Y, x_j) &= H(Y) - \textstyle\sum_x P(x_j = x) H(Y|x_j = x) \\
&= H(Y) - [P(x_j = trumps) H(Y|x_j = trumps) + P(x_j = \neg trumps) H(Y|x_j = \neg trumps)] \\
&= 0.9693 - (0.0669 \times 0.1392 + 0.9331 \times 0.9835) \\
&= 0.04228367
\end{aligned}$$

The mutual information obtained in part B is slightly smaller than part A. This might be because 'trumps' is one of the words in the first layer of the original decision tree. However, it's better for the mutual information to be bigger. Therefore, it's better to keep word 'the' as the first split rather than using word 'trumps'.