CSC411: Assignment #3

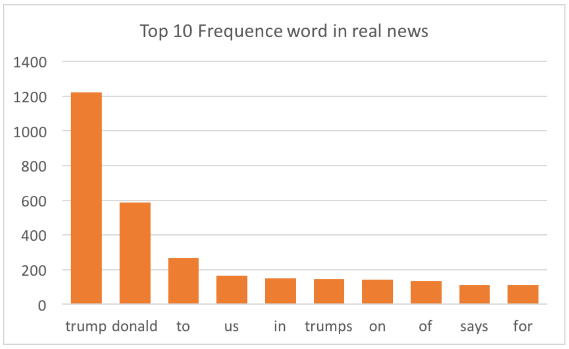
Due on Monday, March 19, 2018

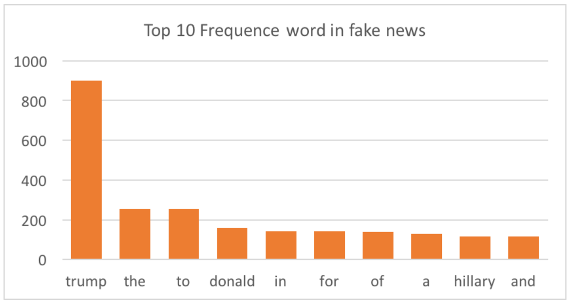
XU WANG, ZILI XIE

March 19, 2018

Part 1

The given source dataset consists of 1298 fake news headlines and 1968 real news headlines. Fake news and real news headlines come from different data source uploaded on the Kaggle website. These two set of data can be further cleaned and stripped, by removing words that are not a part of the headline from the fake news titles and removing special characters from the headlines. Also, we can reduce the noise of data by restricting the range of news headline to be after October 2016 containing the word trump. A cleaned version of fake news and real news headlines data is provided on the website.





3 useful keywords, 'hillary', 'says' and 'us' can be used to determine whether a news is fake or real. The keyword 'hillary' frequently appears in Fake news, so if we see an occurrence of 'hillary' in news headline, then this new is more likely to be a fake one. Similarly, keyword 'says' and 'us' are observed many times in real news, so the news with these keywords are more likely to be real.
So it is feasible to predict real/fake headlines from the words in it, but there are probabilities that it make wrong prediction. Analysis using the complete news content will improve the prediction performance.

```
def build_sets():
    clean_fake = "clean_fake.txt"
    clean_real = "clean_real.txt"

    fake_news = []
    real_news = []

    f_fake = open(clean_fake)
    f_real = open(clean_real)

    lines_fake = f_fake.readlines()
    lines_real = f_real.readlines()

    for line in lines_fake:
        fake_news.append(str.split(line))

    for line in lines_real:
        real_news.append(str.split(line))

    np.random.seed(10)
    np.random.shuffle(fake_news)
    np.random.shuffle(real_news)

    train_fake = int(len(fake_news)*0.7)
    valid_fake = int(len(fake_news)*0.85)
    test_fake  = len(fake_news)

    train_real = int(len(real_news)*0.7)
    valid_real = int(len(real_news)*0.85)
    test_real  = len(real_news)

    train_set = fake_news[0:train_fake]
    train_set = train_set + real_news[0:train_real]

    valid_set = fake_news[train_fake: valid_fake]
    valid_set = valid_set + real_news[train_real: valid_real]

    test_set = fake_news[valid_fake: test_fake]
    test_set = test_set + real_news[valid_real: test_real]

    train_target = [0] * train_fake
    train_target = train_target + ([1] * train_real)

    valid_target = [0] * (valid_fake - train_fake)
    valid_target = valid_target + ([1]* (valid_real - train_real))

    test_target = [0] * (test_fake - valid_fake)
    test_target = test_target + ([1]* (test_real - valid_real))

    return train_set, valid_set, test_set, train_target,
    valid_target, test_target, train_fake, train_real
```

To split the dataset into 70% training, 15% validation, 15% testing. First read all the headlines into two separate lists, from both fake news and real news. Then read the headlines one by one and assign the first 70% to train set, headline from 70% to 85% goes to validation set, rest of them goes to the test set.

Part 2

First we need build the training set, validation set, test set and corresponding training target, validation target, test target. After get the headline from the txt file, we randomly shuffle the headlines, take first 75% as training set, 15% for both validation set and test set.
After we have the set. We construct two dictionaries using the training set. First dict is every words' appear number in the fake news in training set. Another dict is every words' appear number in the real news in training set(Notice: if the word occurs more than once in one head-line, we will only count the number as 1). So we can use these dictionaries to compute the probability of every words using $P(x_i = 1|class) \approx \frac{count(x_i=1,class)+m*\hat{p}}{count(class)+m}$.

$P(x_i = 0|class) = 1 - P(x_i = 1|class)$

Then we can start building the Naive Bayes. For every words in the input headline, we have two situations:
a) The word appears in the training set, then we can compute the probability of $P(word|fake)$ and $P(word|real)$ with the m and $\hat{p}$
b) The word didn't appear in the training set, we can still compute the probability since we have m and $\hat{p}$
For the words didn't appear in the headline but appears in the training set, we compute the probability for them as (1- $P(word = 1|fake)$) and (1-$P(word = 1|real)$)
For general,given a headline, we need to compute the exact probability for the head line being fake/real and choose the one with prob¿0.5 to classify the headline:
$P(fake|w_1, w_2...., w_n) = \frac{P(w_1,w_2...w_n|fake)*P(fake)}{P(w_1,w_2,..w_n)}$

$P(real|w_1, w_2...., w_n) = \frac{P(w_1,w_2...w_n|real)*P(real)}{P(w_1,w_2,..w_n)}$

but since the denominator are the same, so we only compute:
$P(w_1, w_2...w_n|fake) * P(fake) = P(w_1|fake) * P(w_2|fake)...P(w_n|fake) * P(fake)$
$P(w_1, w_2...w_n|real) * P(real) = P(w_1|real) * P(w_2|real)...P(w_n|reap) * P(real)$
And we only need to classify the headline to which case is bigger.
To compute $P(w_1, w_2...w_n|fake)$ and $P(w_1, w_2...w_n|real) * P(real)$ we have:
$P(w_1, w_2...w_n|fake) * P(fake) = \prod_{i=1}^{i_n} P(w_i|fake) * P(fake)$

$P(w_1, w_2...w_n|real) * P(real) = \prod_{i=1}^{i_n} P(w_i|real) * P(real)$

In the above steps we involved computing products of many small numbers, so use the fact given in handout:
$P(w_1, w_2...w_n|fake) = exp[\sum_i log(P(x_i|fake))]$

$P(w_1, w_2...w_n|real) = exp[\sum_i log(P(x_i|real))]$

Our Naive Bayes built as following:

```
def naive_bayes(headline, m, p_hat, dict_fake_appear,
dict_real_appear,pfake,preal,f_count, r_count):
    line = list(set(headline))
    line.sort(key=headline.index)
    probs_fake = []
    probs_real = []
    for word in line:
        if word not in dict_fake_appear:
            prob_fake = (float(m* p_hat)/float(f_count + m))
        else:
            prob_fake = (float(dict_fake_appear[word] + m* p_hat)/float(f_count + m))

        if word not in dict_real_appear:
            prob_real = (float(m* p_hat)/float(r_count + m))
        else:
            prob_real = (float(dict_real_appear[word] + m* p_hat)/float(r_count + m))

        probs_fake.append(math.log(prob_fake))
        probs_real.append(math.log(prob_real))

    for word in dict_fake_appear:
        if word not in line:
            prob_fake = 1 - (float(dict_fake_appear[word] + m* p_hat)/float(f_count + m))
        probs_fake.append(math.log(prob_fake))
    for word in dict_real_appear:
        if word not in line:
            prob_real = 1 - (float(dict_real_appear[word] + m* p_hat)/float(r_count + m))
        probs_real.append(math.log(prob_real))

    p_fake = math.exp(sum(probs_fake))*pfake
    p_real = math.exp(sum(probs_real))*preal

    if p_fake > p_real:
        return 0
    else:
        return 1
```

For prior m and $\hat{p}$.
We tuning them using the validation set, we tried ranged m from 0-10 and $\hat{p}$ from 0-1, cause many of the words' appearance are not very high, so we do not want m to influence much on the prob.After tried different values, we found the m= 2 , $\hat{p} = 0.4$ produce the best performance on validation set. So we applied this to test set and training set.
So the final result we get is :
Train performance: 0.944
Valid performance: 0.853
Test performance : 0.862

Part 3

part3(a)

In order to find the word that presence influence most strongly predicts that the news is real or fake, we need compute the probability $P(fake|word)$ and $P(real|word)$ for every word, then we can pick top 10 probability for both fake and real.We have:

$P(fake|word) = \frac{P(word|fake)*P(fake)}{P(word)}$

$P(real|word) = \frac{P(word|real)*P(real)}{P(word)}$

$P(word|fake)$ and $P(word|real)$ is what we computed in part2 using m and $\hat{p}$

$P(word) = P(word|class = fake) * P(class = fake) + P(word|class = real) * P(class = real)$

Then we can get probability $P(fake|word)$ and $P(real|word)$ for every word, We pick 10 highest from both fake and real. Then we can get the words we need.

```
for word in headline:
    if word not in word_absence_fake:
        if word not in dict_fake_appear:
            prob_noword_fake = 1 - ((float(m* p_hat)/float(train_fake + m)))
            prob_word_fake = ((float(m* p_hat)/float(train_fake + m)))
        else:
            prob_noword_fake = 1 - (float(dict_fake_appear[word] + m* p_hat)/float(train_fake + m))
            prob_word_fake = (float(dict_fake_appear[word] + m* p_hat)/float(train_fake + m))
        if word not in dict_real_appear:
            prob_noword_real = 1 - (float(m* p_hat)/float(train_real + m))
            prob_word_real = (float(m* p_hat)/float(train_real + m))
        else:
            prob_noword_real = 1 - (float(dict_real_appear[word] + m* p_hat)/float(train_real + m))
            prob_word_real = (float(dict_real_appear[word] + m* p_hat)/float(train_real + m))

        prob_word = prob_word_fake*pfake + prob_word_real*preal
        prob_noword = 1 - prob_word

        prob_word_fake_ = float(prob_word_fake * pfake)/float(prob_word)
        prob_word_real_ = float(prob_word_real * preal)/float(prob_word)
        prob_noword_fake_ = float(prob_noword_fake * pfake)/float(prob_noword)
        prob_noword_real_ = float(prob_noword_real * preal)/float(prob_noword)
```

a) presence most strongly predicts that the news is real:
('trumps', 0.99486), ('north', 0.98636), ('korea', 0.98562), ('turnbull', 0.98078), ('travel', 0.98031), ('ban', 0.96706), ('australia', 0.96613), ('paris', 0.95921), ('debate', 0.95184), ('flynn', 0.94875)

b) presence most strongly predicts that the news is fake:
('breaking', 0.96114), ('soros', 0.94517), ('woman', 0.94517), ('daily', 0.94113), ('party', 0.94113), ('black', 0.92678), ('supporter', 0.92678), ('my', 0.92448), ('steal', 0.92448), ('duke', 0.92448)

For the absence, it's similar to the presence. We compute the probability $P(fake|\neg word)$ and $P(real|\neg word)$ for every word.We have:

$P(fake|\neg word) = \frac{P(\neg word|fake)*P(fake)}{P(\neg word)}$

$P(fake|\neg word) = \frac{(1-P(word|fake))*p(fake)}{(1-P(word))}$

$P(real|\neg word) = \frac{P(\neg word|real)*P(real)}{P(\neg word)}$
$P(real|\neg word) = \frac{(1-P(word|real))*p(real)}{(1-P(word))}$
So we can get the probability $P(fake|\neg word)$ and $P(real|\neg word)$ for every words, we still pick 10 highest from both fake and real. And we get the words we need.

c) absence most strongly predicts that the news is real:
('trump', 0.94636), ('the', 0.66028), ('to', 0.62656), ('hillary', 0.62636), ('a', 0.62474), ('of', 0.6231), ('and', 0.62073), ('is', 0.62065), ('for', 0.62042), ('in', 0.61915)

d) absence most strongly predicts that the news is fake:
('donald', 0.48566), ('trumps', 0.42599), ('us', 0.41996), ('says', 0.40976), ('north', 0.40746), ('korea', 0.40691), ('ban', 0.40628), ('turnbull', 0.40438), ('travel', 0.4042), ('comey', 0.40179)

We can see from the above words and count number. The probability of $P(fake|word)$ and $P(real|word)$ are higher than $P(fake|\neg word)$ and $P(real|\neg word)$. So in general, the presence will influnce more on claasifiying the headline is real or fake.

part3(b)
After removing the STOP_WORDS, we can get the following words.
a) presence most strongly predicts that the news is real:
('trumps', 0.99486), ('north', 0.98636), ('korea', 0.98562), ('turnbull', 0.98078), ('travel', 0.98031), ('ban', 0.96706), ('australia', 0.96613), ('paris', 0.95921), ('debate', 0.95184), ('flynn', 0.94875), ('refugee', 0.94524)

b) presence most strongly predicts that the news is fake:
('breaking', 0.96114), ('woman', 0.94517), ('soros', 0.94517), ('daily', 0.94113), ('party', 0.94113), ('supporter', 0.92678), ('black', 0.92678), ('steal', 0.92448), ('duke', 0.92448), ('voting', 0.9081)

c) absence most strongly predicts that the news is real:
('trump', 0.94636), ('hillary', 0.62636), ('clinton', 0.61885), ('just', 0.61475), ('america', 0.61018), ('win', 0.61018), ('supporters', 0.60961), ('watch', 0.60928), ('obama', 0.60905), ('campaign', 0.60897)

d)absence most strongly predicts that the news is fake:
('donald', 0.48566), ('trumps', 0.42599), ('says', 0.40976), ('north', 0.40746), ('korea', 0.40691), ('ban', 0.40628), ('turnbull', 0.40438), ('travel', 0.4042), ('comey', 0.40179), ('china', 0.40143)

part3(c)
Remove stop words:
It make sense to remove stop words when interpreting the model is because these stop words are useless when determining the headline, they are non-sense words and can not illustrate any difference between the real news and fake news. So during the training, it will add more noise to our training set, also because some stop words' appearance may higher than the keywords we need to classify the news, so these non-sense stop words will become the most important feature. In that case, it is easily to cause overfitting because they do not help classify when we apply the model to test set but the model can do well in training set.

Keep stop words:
But it also make sense to keep stop words.Refer to what professor explained in the Piazza, sometimes the stop words "can" help classify the fake and real, not by their meaning, but due to the different edit behaviors for real news editor and fake news editor.

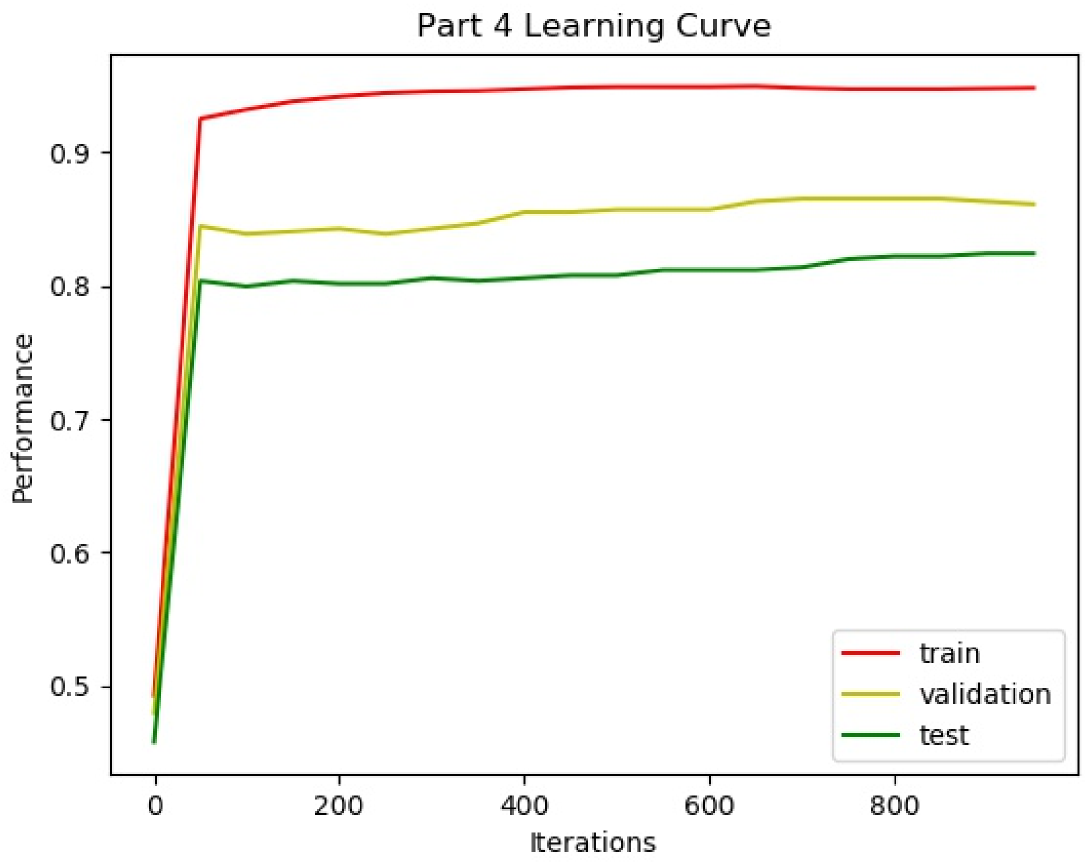
Part 4

In the fourth Part of project 3, we will use PyTorch to build a logistic regression model similar to the one we've built in project 2. In this part, I wrapped the model in logistic_regression function for the convenience of using in the following function.

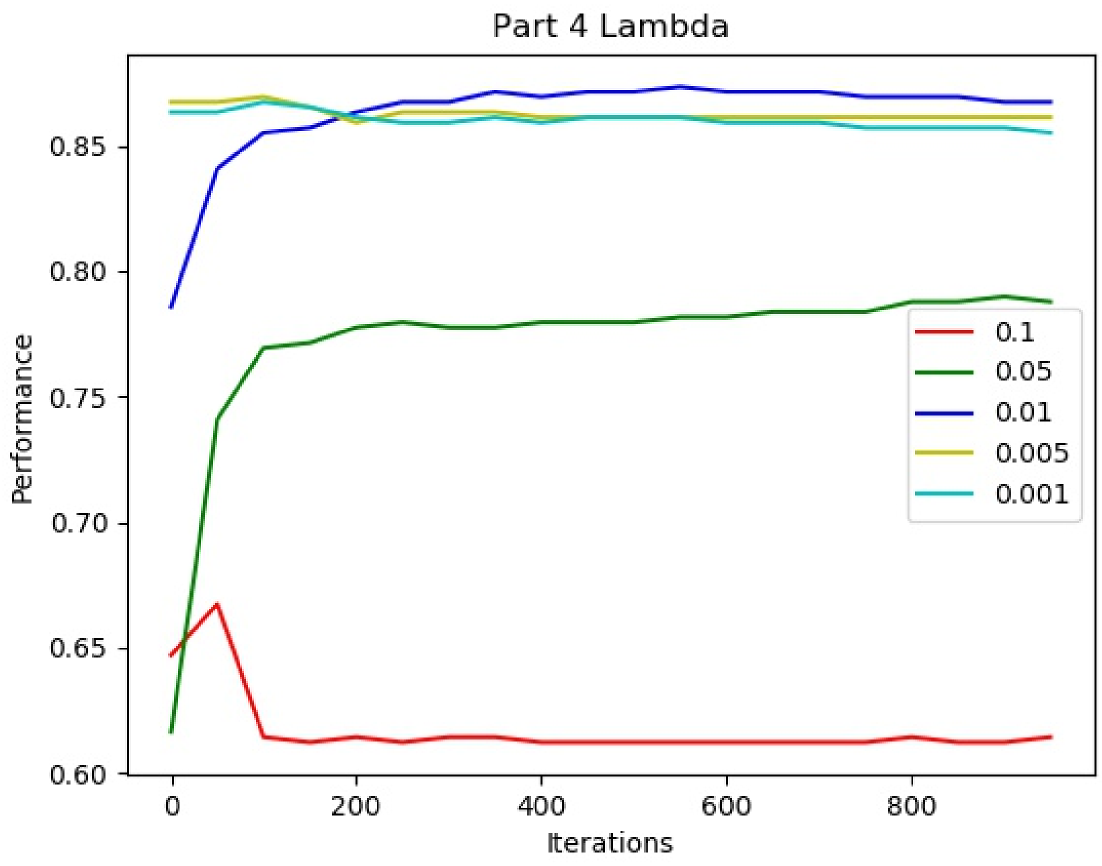
Listing 1: Part4 model

```
def logistic_regression(X, Y_class):
    model = torch.nn.Sequential(
        torch.nn.Linear(X, Y_class),
        torch.nn.Softmax()
    )
    return model
```

In the model above, the X represents the total number of unique word from the complete data set. Y_class represents the number of classes we need to classify, which in this question equals 2. we will also use the optim.Adam optimizer and CrossEntropyLoss as loss function, just like what we done in project 2. What is different is that we added an L2 regularization term to the loss to prevent overfitting and the fast speed of performance on training set reaching the top. The plot of learning curve for this part is shown below. The red line present the performances of model on training set as iteration changes. The yellow line is on validation set and green line is the performance result on test set.



The performance of training set is the highest among three sets, with its learning curve reaches 95% and became stable after the 100th iteration. Validation set and test set have similar curves but reached their plateau at respectively 85% and 80%
As for selecting parameters for this model, learning rate is set to be 6e-4, and the regulation lambda is set to be 0.01. An experiment can be conducted to find the best regulation lambda by comparing the performance curve on validation set of some candidates lambda values.



Obviously is that $\lambda = 0.01$ has the highest accuracy on validation set of more than 85.0%. The curve for $\lambda = 0.01$ has a slightly increasing trend while the curves for 0.001 and 0.005 have decreasing tendency which are not as good as 0.01.
In conclusion, the logistic regression model achieved a final training/validation/test accuracy of 94.75%/86.12%/82.41%.

Part 5

For the Naive Bayes: we use the log-odds of class=real:

$log\frac{P(y=real|x_1,...x_p)}{P(y=fake|x_1,...x_p)} = log\frac{P(y=real)}{P(y=fake)} + \sum_j log\frac{P(x_j|y=real)}{P(x_j|y=fake)}$

$log\frac{P(y=real|x_1,...x_p)}{P(y=real|x_1,...x_p)} = log\frac{P(y=fake)}{P(y=fake)} + \sum_j [log\frac{p(x_j=1|real)}{p(x_j=1|fake)} - log\frac{p(x_j=0|real)}{p(x_j=0|fake)}]x_j + \sum_j log\frac{p(x_j=0|real)}{p(x_j=0|fake)}$
(since x_i can be either 0, 1. so we can write in this formula)

This can be write as $\beta_0 + \sum_j \beta_j x_j$

Compare to the formula:
$\theta_0 + \theta_1 I_1(x_1) + \theta_2 I_2(x_2) + ...\theta_k I_k(x_k)$
we have:
$\theta_0 = log\frac{P(y=real)}{P(y=real)} + \sum_j log\frac{p(x_j=0|real)}{p(x_j=0|fake)}$

$\theta_i = [log\frac{p(x_j=1|real)}{p(x_j=1|fake)} - log\frac{p(x_j=0|real)}{p(x_j=0|fake)}]$

$I_j(x) = x_i$ weather 0 or 1

And the threshold here is equal to 0 since we are computing log-odds. So if $\theta_0 + \theta_1 I_1(x_1) + \theta_2 I_2(x_2) + ...\theta_k I_k(x_k) > 0$ we classify the headline as real news. Other wise we classify it as fake news.

For the Logistic Regression:
We use sigmoid function in the Logistic Regression. So we can write the function as:

$\frac{1}{1+exp(-\theta^T X)} > 0.5$

$\theta^T X > 0$

$\theta_0 + \theta_1 I_1(x_1) + \theta_2 I_2(x_2) + ...\theta_k I_k(x_k) > 0$
So the threshold here is equal to 0
θ here is the weight after training. And $I_j(x) = x_i$ weather 0 or 1 to indicate if the words appears or not.

Part 6

part6(a) (b)

max thetas include stopwords

1 : trumps
2 : debate
3 : voted
4 : tapping
5 : race
6 : where
7 : keating
8 : still
9 : tax
10 : says

min thetas include stopwords

1 : information
2 : victory
3 : build
4 : predicted
5 : breaking
6 : illegal
7 : stunning
8 : go
9 : veterans
10 : already

max thetas exclude stopwords

1 : trumps
2 : debate
3 : voted
4 : tapping
5 : race
6 : keating
7 : tax
8 : says
9 : head
10 : comey

min thetas exclude stopwords

1 : information
2 : victory
3 : build
4 : predicted
5 : breaking
6 : illegal
7 : stunning
8 : veterans
9 : watch
10 : yearns

Part 6(a) contains some similar words comparing to Part 3(a) such as words like trumps, debate. while words like breaking appear in part3(a) in the presence word that the news is fake. So we can draw a conclusion that stop-words in both lists in Parts 6 and 3 have no effect on the top 10 lists.

part6(c)

Generally speaking, using the magnitude of the logistic regression parameters to indicate importance of a feature is a bad idea since in general the features are not normalized. We tuned the magnitude to find the minimal cost, but it the features are not normalized, there are situations that one feature is fairly larger than other features, so in order to decrees the cost, our logistic regression will minimal the weight for this feature, and turns out this feature will be less important then it should be.

However it's reasonable to use magnitude in this problem because our features are either 0 or 1, so there is no chance that one feature will be way too larger than others, so we can say the features are roughly normalized.

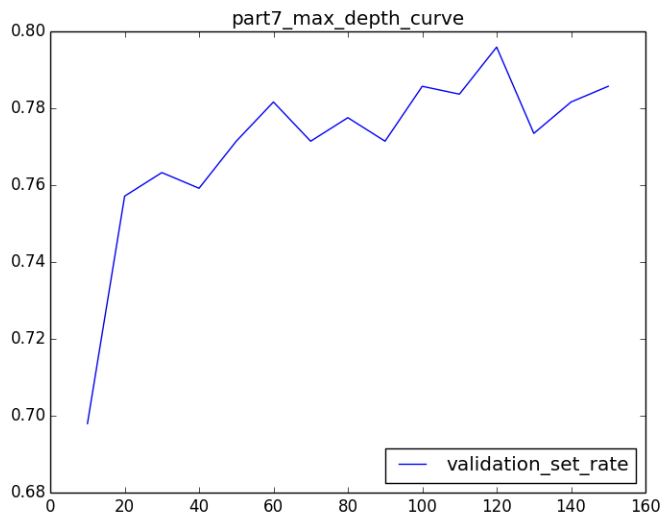
Part 7

part7(a)
First we build the decision tree using the DecisionTreeClassifier, there are some parameters we using that are non-default values:
criterion: We changed criterion to ’entropy’, cause for the following part, we need to compute the mutual information gain, so the criteria ’entropy’ which if for the information gain, can help us better illustration the part8.

random_state: We set the random_state, in order to keep the run result for the decision tree to be the same every time.Because the problem of learning an optimal decision tree is known to be NP-complete problem.

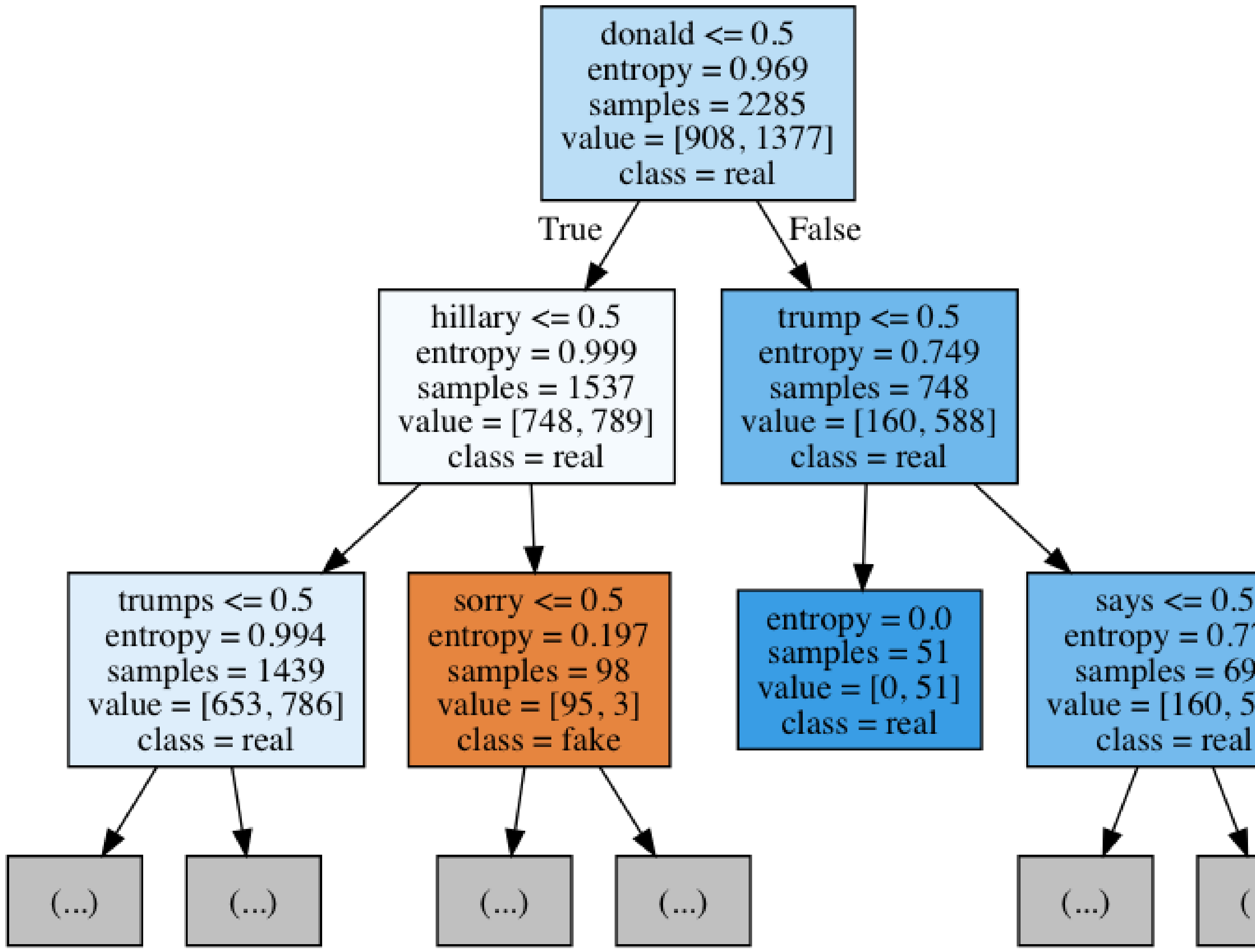
min_samples_leaf: We slightly increased the min_samples_leaf to 4, so when train the decision tree, when the node has less than 4 samples, we will cut off this node.

Then we tried the max_depth from 10 to 150 within interval 10. We plot the curve between max_depth and the accurate of validation set. From the curve about, we can see that at beginning the validation performance increasing with the increase of the max_depth, but after max_depth equal to 100, the performance for the validation set didn’t obviously changed.Which means learning this decision tree may started over-fitting. And we can see from the curve that around max_depth 120 we get the best performance. So we choose the depth_max as 120 to get the testing the training set and test set. We get the performance:
depth is 120 , train: 0.9978118161925602
depth is 120 , valid: 0.7959183673469388
depth is 120 , test : 0.773469387755102



This image display the validation set performance with different max_depth

part7(b)
We visualize the first two split of decision tree using the tool tree.export_graphviz. The visualization is show as below:



This image is the visualization of the first two layer of our DecisionTree

So from the decision tree we can see that the first split is on the feature 'donald', and second layer split is on words 'hillary' and 'trump'. In our previous part, in Naive Bayes and Logistic Regression, we can see this words as well. For example, we can see that the second using the word "hillary", and if "hillary" ¡= 0.5, the class will be real, which means if the "hillary" didn't appear in the headline, it is more likely the headline is real. This matches the result we found in Part3, "hillary" is in the Top 10 words that whose absence most strongly predicts that the news is real. This can indicate that in the different classifier,the features chosen have some similarity.

part7(c)
For Naive Byes:
Train performance: 94.4%
Valid performance: 85.3%
Test performance : 86.2%
For Logistic Regression:
Train performance: 94.75%

Valid performance: 86.12%
Test performance : 82.41%
For Decision Tree:
Train performance: 99.78%
Valid performance: 79.60%
Test performance : 77.34%
So we can tell that Naive Byes and Logistic Regression has fairly the same performance.But the Naive perform better on test set. And Decision Tree has worst performance. Also Decision Tree overfitting most.

Part 8

part8(a)
We will compute the mutual information using the following function

```
def part8(x1,y1,x2,y2,x3,y3):

     prob_x1_x1y1 = float(x1)/float(x1+y1)
     prob_y1_x1y1 = float(y1)/float(x1+y1)
     h_y = (-prob_x1_x1y1*math.log(prob_x1_x1y1)) - (prob_y1_x1y1*math.log(prob_y1_x1y1))

     prob_real = float(x2+y2)/float(x2+y2+x3+y3)
     prob_fake = float(x3+y3)/float(x2+y2+x3+y3)

     prob_2_real = float(x2)/float(x2+y2)
     prob_3_real = float(x3)/float(x3+y3)

     prob_2_fake = float(y2)/float(x2+y2)
     prob_3_fake = float(y3)/float(x3+y3)

     if x2 == 0 or x3 == 0:
          h_y_real = 0
     else:
          h_y_real = (-prob_2_real*math.log(prob_2_real)) - (prob_2_fake*math.log(prob_2_fake))

     if y2 == 0 or y3 == 0:
          h_y_fake = 0
     else:
          h_y_fake = (-prob_3_real*math.log(prob_3_real)) - (prob_3_fake*math.log(prob_3_fake))

     mut_info = h_y - (prob_real*h_y_real + prob_fake* h_y_fake)

     return mut_info
```

In the function above, the input:
x1: The number of real news in the first layer
y1: The number of fake news in the first layer
x2: The number of real news after the first split where the word didn't appear
y2: The number of fake news after the first split where the word didn't appear
x3: The number of real news after the first split where the word appears
y3: The number of fake news after the first split where the word appears
Apply our Decision Tree's visualization above. We have x1= 1377, y1= 908, x2= 789, y2= 748, x3= 588, y3= 160 Then According to the conditional entropy and the formula of mutual information:
$H(Y|class) = P(class = fake) * P(Y|class = fake) + P(class = real) * P(Y|class_real)$
$I(Y; class) = H(Y) - H(Y|class)$
We can get the final mutual information is : 0.0519443092705

part8(b)
For this part, we need slightly change our decision Tree, so that for the first split, it will not always choose the best feature. Then we can get new set of data. For example, we keep the max_features equal to 30, which is very small compare to the amount of the features. Than we train the decision tree again, this time we get a new first split with a new word(feature).

We also tried manually split the tree using other word, get the split result and put into function. Both way works.

We put the new data generated using the word "trump" to our function, we get the mutual information is: 0.0360630856726
Apparently , we can see that the mutual information for the new feature x_j is lower than our original one. That's because, when we build the decision, we choose the criterion as 'entropy', which means the tree will compare the information gain (which is the mutual information in our case) for each features. And choose the feature with the highest information gain. So in part8(a), the split is based on this algorithm with all the features included,so we will get highest mutual information. But in part8(b), we randomly choose another word to split the first layer, the information gain should be lower. This illustrate that our compute function for Part8 is correct.