CSC411 Project 3

Zikun Chen
1001117882

March 19, 2018

Note: the source code fake.py is to be ran in Python 2

Part 1

The data set consists of two text files containing real and fake news headlines respectively. Each headline appears as a single line in the data files. Words in the headline are separated by spaces. Given that preprocessing has been done, the headlines are very clean. For example, there are no special characters or words not part of the headline etc.

I have stored the counts (frequencies) of the unique words into two dictionaries real_train_freq and fake_train_freq.

The prediction is feasible if our assumption holds. For example, words in the headlines are conditionally independent for Naive Bayes model and the ordering of the words does not give information for perdiction (bag of words). If these assumptions fail, our model wouldn't be very accurate.

To give keywords that maybe useful, I've chosen words in training headlines that appear frequently in either of the real or fake set and not the other, excluding common stop words.

3 words from fake news headlines and their frequencies:

```
obama: 40
president: 47
hillary: 107
```

3 words from real news headlines and their frequencies:

```
korea: 54
north: 56
election: 58
```

Part 2

To compute the posterior probability for each unique word in the training set given they are real/fake, we look for its frequency in the two dictionary (frequency of each unique words in real/fake training set), and combined them with the pesudo count m, piror p and number of real/fake news titles in the training set. For example:

$$P(x_j = 1|real) = \frac{count(x_j, real) + m * p}{count(real) * m}$$

$$P(x_j = 0|fake) = 1 - \frac{count(x_j, fake) + m * p}{count(fake) * m}$$

If a word is not in the dictionary, then $count(x_j, y) = 0$, $y \in \{fake, real\}$

Then to compute the maximum a posteriori estimation of the class y given a news headline in the validation/test set using Bayes' rule:

$$\hat{y}_{MAP} = \underset{y}{argmax}[P(y) \prod_{j=1}^{k} P(x_j|y)]$$

where k is number of unique words in the training set, $x_j \in \{0, 1\}$ depending on whether the word appears in the headline, $y \in \{fake, real\}$, $P(real)$ or $P(fake)$ is the proportion of real/fake news headline among all training headlines.

Note, for words in the validation/test set that never appeared in the training set, we ignore them, since they contribute equally to the likelihood of fake and real headlines.

I used grid search to search for the best m from 1 to 5 and the best p from 0.001 to 0.02 (with increment of 0.001). After testing all combinations of m and p on the validation set, the best combination gives the following performance:

Performance:

```
--------- Final Performance ---------

best m: 2
best p: 0.012

training accuracy: 95.32%
validation accuracy: 80.20%
test accuracy: 79.14%
```

Using the fact provided, instead of using product of probabilities in its formula, we can compute $\hat{y}_{MAP}$ with the sum of log probabilities (since the log function is strictly increasing):

$$\hat{y}_{MAP} = \underset{y}{argmax}[\log P(y) + \sum_{j=1}^{k} \log P(x_j|y)]$$

Part 3

(a) 10 words whose presence most strongly predicts that the news is real:

```
flynn: 0.998292398701
refugee: 0.998406057333
week: 0.998406057333
debate: 0.998593316214
paris: 0.998593316214
tax: 0.998860950787
australia: 0.999145469752
travel: 0.99940167544
turnbull: 0.99943015085
korea: 0.999556727862
```

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

```
for: 0.619725220805
is: 0.621190811064
in: 0.622367101304
of: 0.622575832919
and: 0.622958693564
a: 0.625416468348
hillary: 0.628531727652
to: 0.629441624365
the: 0.658155289213
trump: 0.936781609195
```

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

```
gold: 0.997344269751
woman: 0.997344269751
info: 0.997609207846
3: 0.997826080097
liberty: 0.997826080097
u: 0.997826080097
daily: 0.998006879016
go: 0.998006879016
soros: 0.998404867354
breaking: 0.998670369305
```

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

```
australia: 0.402384103013
travel: 0.404533039683
turnbull: 0.404893429309
korea: 0.407069324185
north: 0.407174887892
ban: 0.407723394701
says: 0.410075329567
us: 0.41935483871
trumps: 0.425691514299
donald: 0.483913328956
```

How I did this:

For each word in the training set, for each of the above four lists, we compute:

$$P(y = real|word) = \frac{P(word|y = real)P(y = real)}{P(word|y = real)P(y = real) + P(word|y = fake)P(y = fake)}$$

$$P(y = real|\neg word) = \frac{P(\neg word|y = real)P(y = real)}{P(\neg word|y = real)P(y = real) + P(\neg word|y = fake)P(y = fake)}$$

$$P(y = fake|word) = \frac{P(word|y = fake)P(y = fake)}{P(word|y = real)P(y = real) + P(word|y = fake)P(y = fake)}$$

$$P(y = fake|\neg word) = \frac{P(\neg word|y = fake)P(y = fake)}{P(\neg word|y = real)P(y = real) + P(\neg word|y = fake)P(y = fake)}$$

by looking at the count of the word $P(word|y)$ = count(word, y) and $P(y)$ is the proportion of real/fake news headlines in the training set. And $P(\neg word|y) = 1 - P(word|y)$.

If the count(word, y = real) exists but count(word, y = fake) is 0 (or vice versa), we use the hyper-parameters m and p obtained from Part 2 to estimate:

$$P(word|y) = \frac{m * p}{count(y) + p}$$

Also,

$$P(\neg word|y) = 1 - P(word|y)$$

Influence of Presence vs. Absence: From the above lists, we can see that in general, the influence of presence of words is much stronger than absence of words on predicting whether the headlines are real or fake news. This makes sense because the absence of a word wouldn't give too much information since there are other words out there in the vocabulary not in the headline as well. In fact, a headline only contains around 10 words in the entire unique training vocabulary. The absence of a word shouldn't have a very strong influence on the prediction.

(b) 10 non-stopwords whose presence most strongly predicts that the news is real:

```
flynn: 0.998292398701
refugee: 0.998406057333
week: 0.998406057333
debate: 0.998593316214
paris: 0.998593316214
tax: 0.998860950787
australia: 0.999145469752
travel: 0.99940167544
turnbull: 0.99943015085
korea: 0.999556727862
```

10 non-stopwords whose presence most strongly predicts that the news is fake:

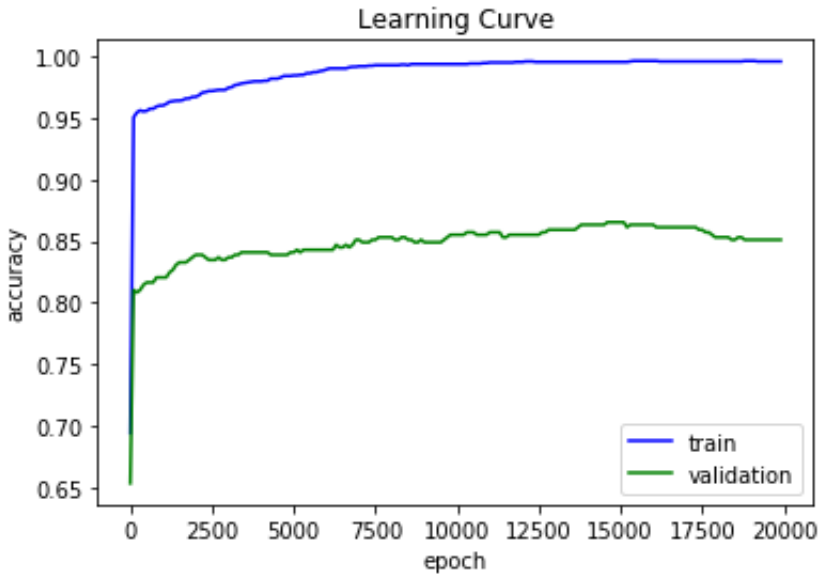
```
writers: 0.997013294956
gold: 0.997344269751
woman: 0.997344269751
info: 0.997609207846
3: 0.997826080097
liberty: 0.997826080097
u: 0.997826080097
daily: 0.998006879016
soros: 0.998404867354
breaking: 0.998670369305
```

(c) It makes sense to exclude stopwords because they can appear in any news headline regardless of it being fake or real. So they don't really gives us more information as to if the title is real.

We want to keep stopwords if we assume that fake headlines are usually not well constructed or have grammatical errors in them. It can be useful in other tasks as well, such as classifying if titles are generated by a robot that is not fully capable of generating titles grammatically correct headlines. In these cases, the count of stopwords would give us information to better predict if a title is real or not.

Part 4

Learning Curve:



Final Performance with the best choice of regularization parameter (1e-4):

```
--------- Final Performance ---------

training accuracy: 99.61%
validation accuracy: 85.10%
test accuracy: 84.66%
```

To select the best regularization parameter, I set the weight_decay parameter of the Adam optimizer in pytorch to various values while holding other hyper-parameters and initilizations fixed and trained the model several times. I select the weight_decay that gives the best validation performance.

The following is a list of performance of each choice of regularization parameter:

```
0
--------- Final Performance ---------

training accuracy: 99.69%
validation accuracy: 83.27%
test accuracy: 84.25%
```

```
0.01
--------- Final Performance ---------

training accuracy: 85.31%
validation accuracy: 82.45%
test accuracy: 82.41%

1e-3
--------- Final Performance ---------

training accuracy: 94.75%
validation accuracy: 84.29%
test accuracy: 84.87%

1e-4
--------- Final Performance ---------

training accuracy: 99.61%
validation accuracy: 85.10%
test accuracy: 84.66%

1e-5
--------- Final Performance ---------

training accuracy: 99.78%
validation accuracy: 83.27%
test accuracy: 83.84%
```

Part 5

Logistic Regression: $thr = 0$
For a particular data point (headline) $x = [x_1, x_2, \cdots, x_k]$:
$x_j = 1$ if the j-th unique word (from the training set) appears in the title and $x_j = 0$ otherwise.

$\theta = [\theta_0, \theta_1, \theta_2, \cdots, \theta_k]$ are weights associated with each features in the input x and θ_0 is the bias.
In the sigmoid function:

$$\sigma(\theta^T x) = \frac{1}{1 + exp(\theta^T x)}$$

- θ_j is the weight associated with the feature x_j
- $I_j(x) = x_j$

Therefore, we classify x to be a real news headline if:

$$\sigma(\theta^T x) > 0.5 \Leftrightarrow \theta^T x > 0$$

$$\Rightarrow \theta^T x = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k = \theta_0 + \theta_1 I_1(x) + \cdots + \theta_k I_k(x) > 0$$

Naive Bayes:
$thr = 0$
$x_j = 1$ if the j-th unique word (from the training set) appears in the headline and $x_j = 0$ otherwise

Look at log-odd of y = real given the input using Naive Bayes:

$$log\frac{P(y = real|x_1, ..., x_k)}{P(y = fake|x_1, ..., x_k)} = log\frac{P(y = real)P(x_1|y = real)...P(x_k|y = real)}{P(y = fake)P(x_1|y = fake)...P(x_k|y = fake)}$$

$$= log\frac{P(y = real)}{P(y = fake)} + \sum_{j=1}^{k} log\frac{P(x_j|y = real)}{P(x_j|y = fake)}$$

Rewrite:

$$log\frac{P(x_j|y = real)}{P(x_j|y = fake)} = \begin{cases} log\frac{P(x_j=1|y=real)}{P(x_j=1|y=fake)}, & \text{if } x_j = 1 \\ log\frac{P(x_j=0|y=real)}{P(x_j=0|y=fake)}, & \text{if } x_j = 0 \end{cases}$$

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

Therefore:

$$log\frac{P(x_j|y = real)}{P(x_j|y = fake)}$$

$$= log\frac{P(y = real)}{P(y = fake)} + \sum_{j=1}^{k} log\frac{P(x_j = 0|y = real)}{P(x_j = 0|y = fake)} + \sum_{j=1}^{k} [log\frac{P(x_j = 1|y = real)}{P(x_j = 1|y = fake)} - log\frac{P(x_j = 0|y = real)}{P(x_j = 0|y = fake)}]x_j$$

$$= \theta_0 + \sum_{j=1}^{k} \theta_j I_j(x) = \theta_0 + \theta_1 I_1(x) + \cdots + \theta_k I_k(x)$$

where

$$\theta_0 = log\frac{P(y = real)}{P(y = fake)} + \sum_{j=1}^{k} log\frac{P(x_j = 0|y = real)}{P(x_j = 0|y = fake)}$$

for $j = 1, \cdots, k$:

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

and

$$I_j(x) = x_j$$

where x is the news headline to be classified.

The headline x is considered real if:

$$\theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k = \theta_0 + \theta_1 I_1(x) + \cdots + \theta_k I_k(x) > 0$$

Part 6

(a) Note the outputs give the index, value and the words associated with each weight θ:

list of top 10 positive θ's:

```
3656 1.73918 speaks
2098 1.7079 where
2010 1.66881 trumps
169 1.65712 us
3086 1.6279 debate
3015 1.62512 tax
3005 1.6036 turnbull
1923 1.58644 comey
7 1.57027 talks
3033 1.55877 australia
```

list of top 10 negative θ's:

```
640 -1.69182 happened
109 -1.69817 breaking
1249 -1.70293 alt
176 -1.70348 they
81 -1.71521 watch
20 -1.7256 won
13 -1.73538 elect
1104 -1.74098 go
123 -1.75051 israeli
1518 -1.76322 bored
```

Like the first list in part 3 a), we can see that in the above list for positive θ, the country "australia" and its prime minister "turnball" appear again in the positive list. Words "debate" and "tax" are in the list as well. This suggest that they are important words in terms of predicting real headlines.

And the word "breaking" seems to be an important word for fake news prediction since it appeared in both parts, which makes sense since fake news tend to lure people with big words like "breaking".

However, other words are not exactly the same as those in part 3, especially for words influencing fake news predictions. But the words here in the negative list and in part 3 do seem to provoke curiosity of the reader.

(b) Ignoring all stop words, we get:

list of top 10 positive θ's:

```
3656 1.73918 speaks
2010 1.66881 trumps
3086 1.6279 debate
3015 1.62512 tax
3005 1.6036 turnbull
1923 1.58644 comey
7 1.57027 talks
3033 1.55877 australia
3954 1.55103 vandalised
2923 1.5454 comments
```

list of top 10 negative θ's:

```
202 -1.65793 victory
892 -1.66473 daily
640 -1.69182 happened
109 -1.69817 breaking
1249 -1.70293 alt
81 -1.71521 watch
20 -1.7256 won
13 -1.73538 elect
123 -1.75051 israeli
1518 -1.76322 bored
```

Similarly, four words in the list of positive θ are the same as in part 3 b). And "daily" and "breaking" appeared in both parts for fake news predictions. Again, the words here in the negative list and in part 3 b) do seem to provoke curiosity of the reader. What is also interesting in the positive list is that some of them show the same theme or action like "speaks", "debate", "talks" and "comments".

(c) In general, using the magnitude of the logistic regression parameters to indicate importance of a feature is a bad idea. Because an outlier in the training set with one feature being very large will influence the associated weights to grow by a lot. This will result in large magnitude in that weight, which seemingly shows the weight is very important but in reality it was large because of the outlier.

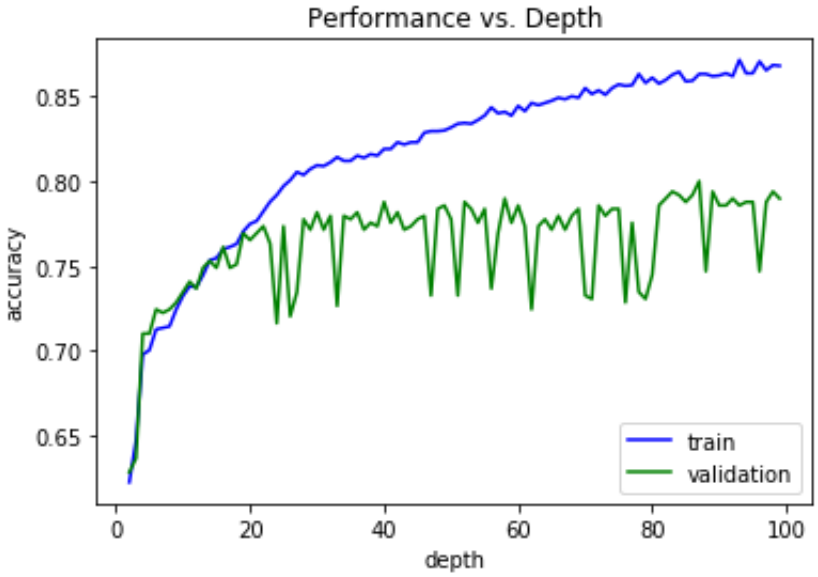
It is reasonable to use the magnitude in this problem because all the features are binary, either a word exists (1) or does not exist (0) in a headline. So there won't be outliers with large features influencing the weights' magnitude.

Part 7

(a) To choose the best-performing decision tree, I used for loop to grow trees with max_depth from 2 to 100, and chose the depth with the highest validation performance.

I chose criterion to be 'entropy', because that is the heuristic that we learned from class. In addition, I kept min_sample_split to 100. This will ensure that we don't split at a node where there are less than 100 samples since we do not what the tree to overfit too much (each leaf end up having too few samples).

Performance vs. Depth:



Performance of the Best Tree (max_depth = 87):

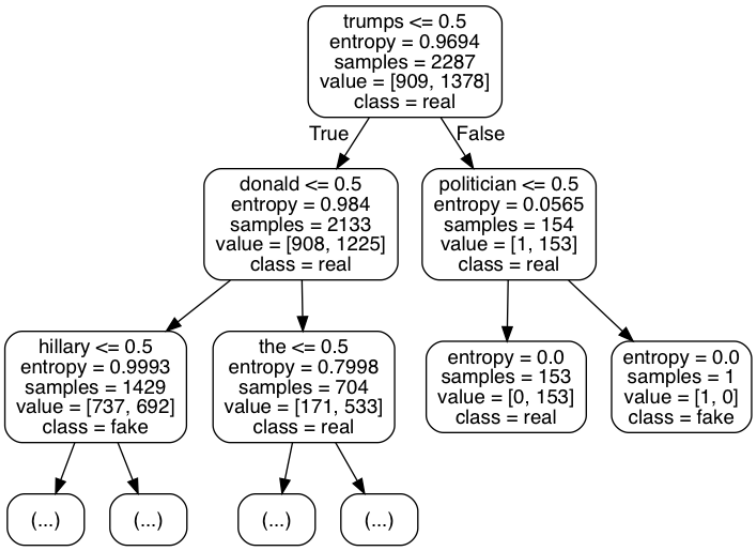
```
--------- Best Performance ---------

training accuracy: 86.31%
validation accuracy: 80.00%
test accuracy: 76.48%
```

(b) **Code for Tree Visualization:**

```
def show_tree(tree, features, depth, path):
    f = StringIO.StringIO()
    export_graphviz(tree, out_file = f, max_depth = depth, rounded = True,
                    feature_names = features, class_names = ['fake', 'real'])
```

```
pydotplus.graph_from_dot_data(f.getvalue()).write_png(path)
img = misc.imread(path)
plt.imshow(img)
```

Visualization:



The words in the top 2 layers are "trumps", "donald", "politician", "hillary" and "trumps".

Interestingly, The decision tree seems to first pick words that are in the two absence lists in part 3 a) to split on. "trumps" and "donald" are the top 2 words whose absence most strongly predict that the news is fake, and "hillary" and "the" also rank high in the list of words whose absence most strongly predict that the news is real. On the other hand, compared to logistic regression and the magnitude of the weights, the words that the decision tree picked to split on first do not correspond to top positive/negative θ's in part 6 a) except "trumps".

(c) **Summary:**

	Naive Bayes	Logistic Regression	Decision Tree
Training Performance	95.32%	99.61%	86.31%
Validation Performance	80.20%	85.10%	80.00%
Testing Performance	79.14%	84.66%	76.48%
Validation/Training	0.84	0.85	0.93

Logistic regression performed the best and desicion tree performed the worst. In terms of overfitting, Naive Bayes overfits the most based on the ratio of validation performance over training performance. On the other hand, the decision tree does not overfit very much.

Part 8

Functions:

```
# entropy calculation given number of fake/real news healines in a node
def H(num_fake, num_real):
    num = float(num_fake + num_real)
    p_fake = num_fake/num
    p_real = num_real/num
    return -(p_fake*math.log(p_fake, 2)
             + p_real*math.log(p_real, 2))

# I(Y; xj) = H(Y) - H(Y | xj)
def mutual_info(hy, p_word, p_noword, hy_word, hy_noword):
    hy_xj = p_word * hy_word + p_noword * hy_noword
    return hy - hy_xj
```

(a) **Code:**

```
# 8 a)  mutual information when splitting on word "trumps"
print '\nmutual information when splitting on "trump":\n'
print mutual_info(H(909, 1378), 1378/2287., 909/2287., H(1, 153), H(908, 1225))
```

Ouput:

```
mutual information when splitting on "trump":
0.544281784985
```

(b) **Code:**

```
# 8 b)  mutual information when splitting on word "hillary"
word_i = np.argwhere(train_unique_words == "hillary")

train_set = train_x[train_idx]
labels = train_y[train_idx]

fake_i = np.argwhere(train_set[:,word_i] == 0)[:,0]
real_i = np.argwhere(train_set[:,word_i] != 0)[:,0]

# when the word not exists
num_real_0 = np.count_nonzero(labels[real_i])
num_fake_0 = len(labels[real_i]) - num_real_0

# when the word exists
num_real_1 = np.count_nonzero(labels[fake_i])
num_fake_1 = len(labels[fake_i]) - num_real_1
```

```
print '\nmutual information when splitting on "hillary":\n'
print mutual_info(H(909, 1378), 1378/2287., 909/2287., \
            H(num_fake_0, num_real_0), H(num_fake_1, num_real_1))
```

Ouput:

```
mutual information when splitting on "hillary":
0.203147138442
```