CSC411 Project 3

Ze Jia Erkang Liu

March 2018

Part 1

The dataset consists of txt files named "cleanreal" and "cleanfake". Each line of these documents is a news headline. The training, validation and test sets are arrays of strings, and each entry to the array is a single news headline. Training, validation and test labels are arrays of integers, with 0s indicating fake news and 1s indicating real news. The training set and label each have 2285 entries; the validation set and label each have 491 entries; the test set and test label each have 490 entries.

Three examples:

- "Obama" appears 47 times in real news and 67 times in fake news.
- "Ban" appears 106 times in real news and 20 times in fake news.
- "America" appears 45 times in real news and 92 times in fake news.

By examining the frequency of the appearances of these words, the type of news can be predicted by the classifiers.

Part 2

The goal of the naive bayes is to find $P(fake|w)$ given $P(w|fake)$
The equation to calculate that is given as:

$$P(fake|w_1....w_n) = \frac{P(fake)\prod_{i=1}^{n} P(w_i|fake)}{P(real)\prod_{i=1}^{n} P(w_i|real) + P(fake)\prod_{i=1}^{n} P(w_i|fake)}$$

The prior belief $P(fake)$is calculated as:

```
count_fake = training_label.count(0)
count_real = training_label.count(1)

P_fake = float(count_fake)/float(len(training_label)) #0.4
P_real = float(count_real)/float(len(training_label)) #0.6
```

We then find $\prod_{i=1}^{n} P(w_i|fake)$ by:

```
m = 2
p_hat = 0.4
fake_total_val = sum(fake_frequency_dictionary.values())
# number of words in fake_content
real_total_val = sum(real_frequency_dictionary.values())
# number of real words in real_content
for word, frequency in fake_frequency_dictionary.iteritems():
    P_frequency = (frequency+m*p_hat)/float(fake_total_val + 2)

    if word in test:
        fake_chances.append(P_frequency)
    else:
        fake_chances.append(1. - P_frequency)

for word, frequency in real_frequency_dictionary.iteritems():
    P_frequency = (frequency+m*(1-p_hat))/float(real_total_val + 2)
    if word in test:
        real_chances.append(P_frequency)
    else:
        real_chances.append(1. - P_frequency)
```

The parameter p_hat was tuned to be P(fake), and the parameter m was chosen to be a small integer number because the number of imaginary samples should not impact the actual statistics too much. After trial and error, m was chosen to be 2 for optimal performance.
Directly multiplying all of the $P(w_i|fake)$ will result in underflow. Usingthe given function of small_multiplication can be avoid the problem:

```
def small_multiplication(n):
    log_sum = 0.
    for num in n:
        log_sum += math.log(num)
    exponent = math.exp(log_sum)
    return exponent
P_words_fake = small_multiplication(fake_chances)
```

```
    P_fake_words = P_fake * P_words_fake

    P_words_real = small_multiplication(real_chances)
    P_real_words = P_real * P_words_real
```

The percentage is then regularized:

```
    P = P_fake_words/(P_fake_words + P_real_words)
```

The performance for training set: 95.10%

The performance for validation set: 85.31%

The performance for testing set: 85.97%

Part 3

Part a.

The 10 words whose presence most strongly predicts that the news is real are:

```
trump
donald
to
us
trumps
in
on
of
for
says
```

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

```
manaforts
asian
liar
stinks
whose
whack
skipping
voted
sorry
worth
```

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

```
trump
to
the
in
donald
for
of
a
on
and
```

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

```
pide
four
hath
deri
hating
assembled
founder
dazu
pantano
cashin
```

The words are stored as keys in 2 dictionaries, real frequency and fake frequency, representing the number of appearances of each work in real and fake news headlines. The dictionary can be sorted based on each key's value. The key to the highest value of the real frequency dictionary would be the word whose presence most strongly predicts that the news if real.

Part b.

Without stop words:
The 10 words whose presence most strongly predicts that the news is real are:

```
trump
donald
trumps
says
north
election
korea
clinton
president
ban
```

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

```
manaforts
asian
liar
stinks
whack
skipping
voted
sorry
worth
risk
```

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

```
trump
donald
hillary
clinton
president
election
new
just
america
obama
```

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

```
pide
hath
deri
hating
assembled
founder
dazu
pantano
cashin
teaching
```

Part c.

As can be seen if we compare the 10 words whose presence most strongly predicts that the news is real from section a and from section b, common words such as "in", "on" or "says" is removed from the list since these words would occur the most in any news headline so they convey very little information. Another example is that the 10 words whose presence most strongly predicts that the news is fake also consists of "in" and "on", which further proves the earlier point.

Part 4

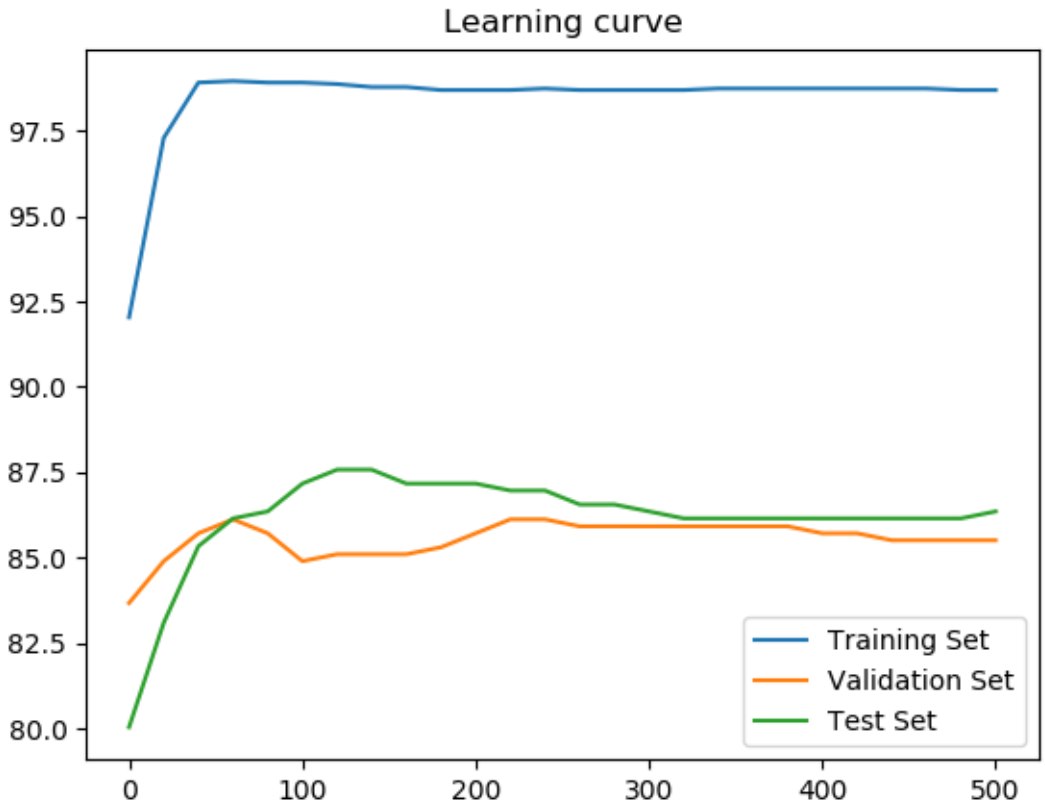

Figure 1: Logistic Regression Learning Curve

Iteration: 0 Training Performance : 91.94748358862145% Validation Performance : 83.06122448979592% Test Performance : 81.05906313645622%
Iteration: 20 Training Performance : 97.24288840262581% Validation Performance : 85.51020408163265% Test Performance : 82.4847250509165%
Iteration: 40 Training Performance : 98.94967177242889% Validation Performance : 85.71428571428571% Test Performance : 85.33604887983707%
Iteration: 60 Training Performance : 98.94967177242889% Validation Performance : 86.12244897959184% Test Performance : 86.15071283095723%
Iteration: 80 Training Performance : 98.86214442013129% Validation Performance : 85.10204081632654% Test Performance : 86.35437881873727%
Iteration: 100 Training Performance : 98.90590809628009% Validation Performance : 84.89795918367346% Test Performance : 86.9653767820774%
Iteration: 120 Training Performance : 98.86214442013129% Validation Performance : 84.89795918367346% Test Performance : 87.37270875763747%

Iteration: 140 Training Performance : 98.7746170678337% Validation Performance : 85.10204081632654% Test Performance : 87.57637474541752%
Iteration: 160 Training Performance : 98.7746170678337% Validation Performance : 85.10204081632654% Test Performance : 87.16904276985743%
Iteration: 180 Training Performance : 98.6870897155361% Validation Performance : 85.3061224489796% Test Performance : 87.16904276985743%
Iteration: 200 Training Performance : 98.6870897155361% Validation Performance : 85.91836734693878% Test Performance : 87.16904276985743%

Logistic regression is implemented with PyTorch. We used L2 regularization instead of L1 regularization. The λ is chosen to be 0.01. A range of different λ, (i.e 0.01, 0.02, 0.03,0.005), was tried, and 0.01 provided the best result. L1 regularization had significantly worse result compared to L2 regularization.

The L2 regularization is implemented as this:

```
    l2_reg = Variable( torch.FloatTensor(1), requires_grad=True)
  for W in model.parameters():
    if l2_reg is None:
      l2_reg = W.norm(2)
    else:
      l2_reg = l2_reg + W.norm(2)
  loss = loss_fn(prediction, y_classes) + reg_lambda * l2_reg
```

We can see that the best result from linear regression is around 160th iteration.
The performance for training set: 98.774%
The performance for validation set: 85.10%
The performance for testing set: 87.16%

Part 5

Logistic Regression:

θ_0 is ~~weight~~ bias

$\theta_1 \dots \theta_k$ are the weights

$I_1 \dots I_k = I(x_1) \dots I(x_k) = x_1 \dots x_k$ are the inputs

each of the k xs represents one of the k features

if a feature ~~exists~~ k exists, $x_k = 1$, otherwise $x_k = 0$

Naive ~~Bayes~~ Bayes:

$$\theta_0 = \log \frac{P(\text{real})}{P(\text{fake})} + \sum_{i}^{i=k} \log \frac{P(x_i = 0 \mid \text{real})}{P(x_i = 0 \mid \text{fake})}$$

for $i = 1 \dots k$

$$\theta_i = \log \frac{P(x_i = 1 \mid \text{real})}{P(x_i = 1 \mid \text{fake})} - \log \frac{P(x_i = 0 \mid \text{real})}{P(x_i = 0 \mid \text{fake})}$$

~~$I_1 \dots I_k$~~

$$I_i(x) = x_i \begin{cases} 0 & \text{if feature DNE} \\ 1 & \text{otherwise} \end{cases}$$

Part 6

a.

top 10 positive weights and words:

```
1: breaking: 2.0245886
2: watch: 1.8602487
3: are: 1.6191616
4. black: 1.5378845
5. that: 1.5378845
6. already: 1.4355259
7. u: 1.3924543
8: just: 1.3632052
9: go: 1.3627362
10: won: 1.3228551
```

top 10 negative weights and words:

```
1: trumps: -3.6532564
2: us: -1.8811134
3: turnbull: -1.6231906
4. australia: -1.6056185
5. tax: -1.5095206
6. comey: -1.423276
7. donald: -1.4153867
8: korea: -1.3168638
9: says: -1.236738
10: north: -1.2028866
```

b.

Without stop words, top 10 positive weights and words:

```
1: hillary: 2.1386209
2: breaking: 2.0245483
3: watch: 1.8602487
4. black: 1.5378845
5. u: 1.3924543
6. just: 1.3632052
7. won: 1.3228551
8: daily: 1.3146912
9: alt: 1.290858
```

```
10: information: 1.2868476
```

top 10 negative weights and words:

```
1: trumps: -3.6532564
2: turnbull: -1.6231906
3. australia: -1.6056185
4. tax: -1.5095206
5. comey: -1.423276
6. donald: -1.4153867
7: korea: -1.3168638
8: says: -1.236738
9: north: -1.2028866
10: decision: -1.1591877
```

Part 7

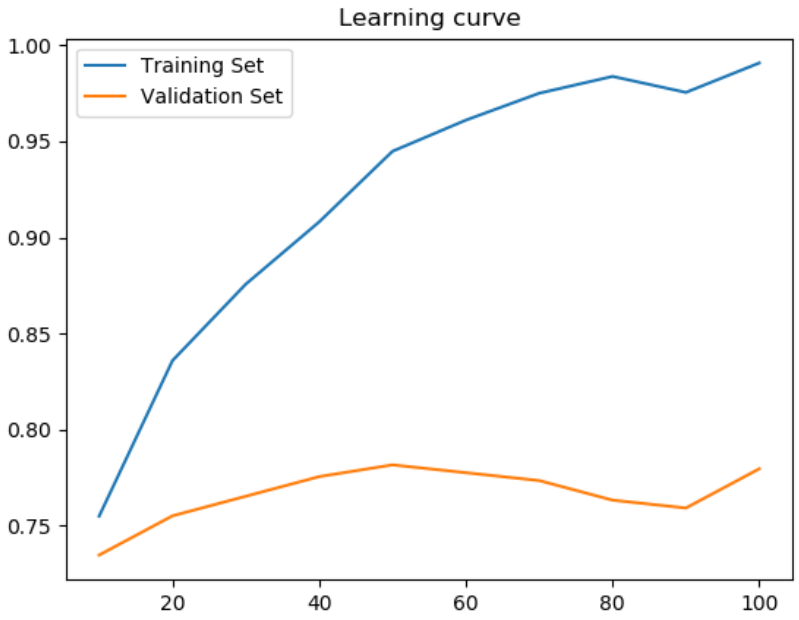

Figure 2: Max depths vs accuracy

As shown in the plot, overfitting starts to happen after max depth of 50, so max depth of 50 was chosen. iteration: 10 train score: 75.62363238512036% validation score: 73.06122448979592%

iteration: 20 train score: 83.54485776805251% validation score: 75.51020408163265%
iteration: 30 train score: 87.527352297593% validation score: 76.73469387755102%
iteration: 40 train score: 92.03501094091904% validation score: 76.53061224489795%
iteration: 50 train score: 94.87964989059081% validation score: 75.71428571428571%
iteration: 60 train score: 95.36105032822758% validation score: 78.16326530612245%
iteration: 70 train score: 96.10503282275711% validation score: 76.12244897959184%
iteration: 80 train score: 97.89934354485777% validation score: 76.53061224489795%
iteration: 90 train score: 98.24945295404814% validation score: 77.75510204081633%
iteration: 100 train score: 97.81181619256017% validation score: 76.32653061224491%

b.

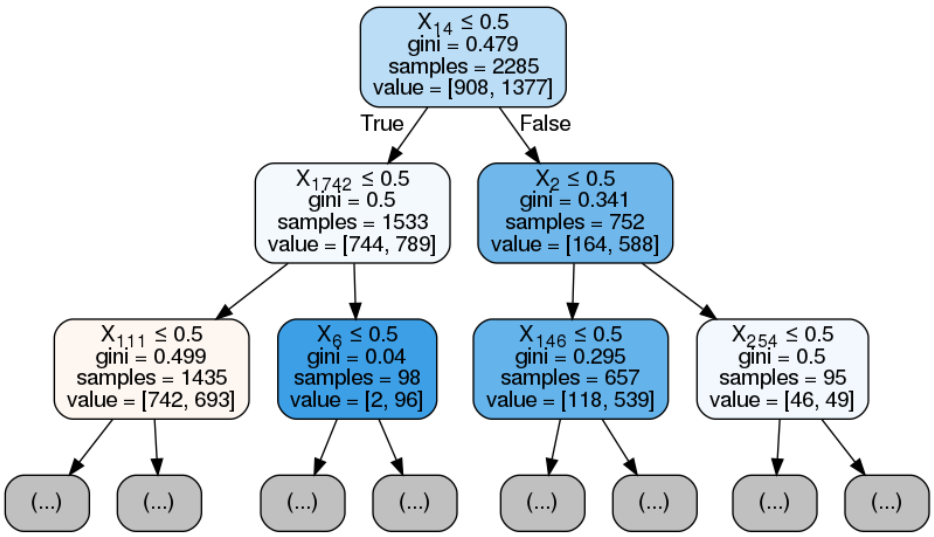

Figure 3: Graphical Representation of Decision Tree

The most significant node is the word "the", which would make sense since it is probably the most common node. Going deeper, the first node if True is "hillary", which is the word with most positive weight, and the first node if False is "trumps", which is is the word with most negative weight. This makes sense because decision tree can be more efficient if the nodes with biggest information gain is at a higher level.

c.

- Naive Bayes: 95.10 % on training set, 85.31% on validation set, 85.97% on test set.
- Linear Regression: 98.69 % on training set, 85.92% on validation set, 87.17% on test set.
- Decision: 94.88 % on training set, 75.71% on validation set.

Out of the three classifiers, linear regression classifier has the best performance on validation set, and decision tree has the worst performance on validation set. Decision tree overfits the most.

Part 8

a.

$$H(Y) = -\frac{908}{2285}\log_2\frac{908}{2285} - \frac{1377}{2285}\log_2\frac{1377}{2285} = 0.9694$$

$$P(\text{True}) = \frac{1533}{2285} \qquad H(\text{True}) = 0.9994$$

$$P(\text{False}) = \frac{752}{2285} \qquad H(\text{False}) = 0.7567$$

$$I = H(Y) - H(Y \mid X_i)$$

$$= H(Y) - P(\text{True})H(\text{True}) - P(\text{False})H(\text{False})$$

$$= 0.04987$$

b.

b. $H(Y) = 0.7567$

$P(\text{True}) = \frac{753}{2285}\ \frac{657}{752}$ $\qquad H(\text{True}) = 0.67921$

$P(\text{False}) = \frac{95}{752}$ $\qquad H(\text{False}) = 0.9993$

$I = 0.03705$

As a comparison, the information gain in part b is 4.987% which is smaller than the information gain from part a, which is 3.705%. This means that the information that the node offers about other nodes is more in the node from part a. Since that node has the largest information gain, it is chosen to be the starting node.