CSC411: Project #1

Due on Monday, March 19, 2018

Izaak Niksan

Part 1)
Upon first glance, one striking difference between the datasets is that the real articles appear to have significantly longer headlines on average. In terms of words that seem to appear more frequently in a specific dataset, the words *hillary*, *russia*, and *victory* immediately stick out. The word *hillary* appears 150 times in fake articles but only 24 times in real articles. The word *russia* appears 45 times in fake articles and 77 times in real articles. The word *victory* appears 48 times in fake articles and 25 times in real articles. The stated goal of the project is likely feasible since some words appear more often in one type than the other.

Part 2)
First, the data was separated into different sets by running the *splitting_datasets.py* script. (For consistency, this was never run again. Thus, please do not run this script as the datasets will change since randomized splitting was employed.) This script parses the provided fake and real headlines and dumps various dictionaries into pickle files for easy access in future parts. The training set comprises 70% of the headlines, the validation set 15%, and the test set 15%. These dictionaries contain words as keys, and the number of times each word appears as values.

In *part2.py*, the Naive Bayes classifier was implemented. The code for the function is seen in Listing 1. As noted in the project handout, it is undesirable to multiply many small floating point values together since approximations in their digital representations eventually lead to very inaccurate results. Instead, when implementing the classifier, a sum of logs was used instead. More specifically, when evaluating

$$P(C|headline) = \frac{P(headline|C)P(C)}{P(headline)} = \frac{P(C)\Pi_i P(word_i|C)}{P(headline)},$$

instead of explicitly computing the term on the right, since the end goal was to determine which of $P(fake|headline)$ or $P(real|headline)$ was larger, it was sufficient to maximize the term on the right. Thus, since exponential functions are strictly increasing or decreasing and the term in the denominator is a constant, the term on the right was replaced with $log(P(headline)) + \sum_i log(P(word_i|C))$.

One additional note is that, as described in the the lecture slides, "virtual examples" were added to the training set to avoid the case where the count of a word is 0. This was done by including two hyperparameters: m and p_hat. Using these, $P(word_i|C)$ was computed as follows:

$$P(word_i|C) = \frac{count(word_i, C) + m * p\_hat}{count(C) + m}$$

if $word_i$ appears in the headline, and otherwise as

$$P(word_i|C) = 1 - \frac{count(word_i, C) + m * p\_hat}{count(C) + m}$$

The ideal values for m and p_hat were determined to be 1 and 0.35 respectively. They were optimized using the validation set; m was varied through integer values between 1 and 10, and p_hat was varied through multiples of 0.05 between 0.05 and 1. The code for this optimization is also found in *part2.py*.

The performance of the classifier on the training set and test set were 96.4% and 84.9% respectively.

Listing 1: Naive Bayes Classifier

```
def naive_bayes_istrue(headline,real,fake,real_count,fake_count,m,p_hat):
    '''
    Inputs:
        headline:    New headline which will be classified as real or fake. It
```

```
                    is assumed that the headline is already cleanly processed.
        real:         Dictionary with keys as words which appear in real
                      headlines and values as the number of times the words
                      appear.
        fake:         Same type of dictionary but for fake headlines.
        real_count:   Number of real headlines.
        fake_count:   Number of fake headlines.
        m:            Number of virtual examples added to the sets.
        p_hat:        Prior probability.

    Output:
        Boolean value, with true being true and false being fake
    '''

    line=headline
    line=line.rstrip('\n')
    temp=line.split(' ') # words contains all the words in the headline

    #remove any duplicated words in the headline:
    words=[]
    [words.append(item) for item in temp if item not in words]

    preal=real_count/(real_count+fake_count) #p(real)
    pfake=fake_count/(real_count+fake_count) #p(fake)
    real_prob=0 #Probability that headline is real
    fake_prob=0 #Probability that headline is fake

    #First calculate the probability that headline is real
    for key in real:
        if key in words:
            real_prob+=log((real[key] + m*p_hat)/(real_count+m))
        else:
            real_prob+=log(1-(real[key] + m*p_hat)/(real_count+m))
    real_prob+=log(preal)

    #Now calculate the probability that headline is fake
    for key in fake:
        if key in words:
            fake_prob+=log((fake[key]+m*p_hat)/(fake_count+m))
        else:
            fake_prob+=log(1-(fake[key]+m*p_hat)/(fake_count+m))

    fake_prob+=log(pfake)

    #At this point, we have been summing the log probabilities. We could
    #exponentiate then divide by the normalization factor to get the actual
    #probabilites, but instead we just return the larger of the two values
    #since we only care about relative sizes.

    if fake_prob>real_prob:
        return False
    else:
        return True
```

Part 3)

a) The presence of words which appear most frequently in real headlines contribute most towards predicting that the headline is real. Conversely, the absence of words which appear least frequently in real headlines contribute most towards predicting that he headline is real. The same logic applies to fake headlines.

The basis for this is the conditional probability formula, specifically the term in the numerator of Baye's Rule:

$$P(headline|C) = \Pi_i P(word_i|C)$$

For a given class, if the word appears in the headline then the factor in the multiplication corresponding to this word is calculated as the ratio of the number of times this word appears in headlines of this class to the number of total headlines of this class. Maximizing this ratio makes a larger contribution to increasing the overall probability. Similarly, if the word does not appear in the headline, the term instead becomes this same ratio, but subtracted from 1. Therefore, minimizing the ratio will lead to increasing the overall probability.

The 10 words whose presence most strongly predicts that the news is real are: *trump, donald, to, us, in, trumps, on, of, says,* and *for.*

The 10 words whose absence most strongly predicts that the news is real are: *truths, pedo, ring, guerrilla, economist, avoid, scissors, ribbon, c,* and *hypocrisy.*

The 10 words whose presence most strongly predicts that the news is fake are: *trump, to, the, donald, for, in, of, and, a,* and *on.*

The 10 words whose absence most strongly predicts that the news is fake are: *avocado, grattan, date, jennett, backwards, jordan, vow, personnel, industrials,* and *denuclearisation.*

*b) *It is assumed that this question is asking for the words whose presence most affects the probability**

The 10 non-stopwords whose presence most strongly predicts that the news is real are: *trump, donald, trumps, says, north, korea, election, clinton, ban,* and *president.*

The 10 non-stopwords whose presence most strongly predicts that the news is fake are: *trump, donald, hillary, clinton, election, just, president, new, america,* and *obama.*

The explanation of why prevalence leads to increased probability is the same as what was described in part a).

c) It usually makes sense to remove stopwords as they generally do not provide any meaningful information to the classifier since they are typically not relevant to the classification problem. In fact, having lots of stopwords in the datasets may actually confuse the classifier. However, in the rare case where one determines that the stopwords are actually relevant to the problem, then it may be useful to keep them in the datasets. One such example may be in the classification of spam emails; if it turns out to be the case that spam emails use many stopwords, since these emails generally do not have much meaningful content, then not removing them may be a wise decision.

Part 4)

In order to formulate this as a logistic regression classification problem, first the input vector $v \epsilon n x m$ (for m headlines and n inputs) was set up such that $v[k][z] = 1$ when word k appeared in headline z, else the value was 0.

As before, the headlines were cleaned up such that they became arrays of words, with spaces and newline characters removed. Some code from project 2 was reused. The inputs consisted of the words in the *training set* only, since words which the classifier has never seen before cannot be used in any meaningful way. The output of the classifier is $y \epsilon 2 x m$, such that $y[0][k] = 1$ and $y[1][k] = 0$ for a real headline, and the reverse for a fake one. In other words, it employs one-hot encoding. When an input is fed into the classifier, the index whose value is closest to 1 is taken as the classifier's guess.

For the gradient descent, alpha was 0.001, lambda was 0.8, and MAX_ITERS was 10 000. As recommended on piazza, L2 regularization was used. It is noted that the value of lambda did not seem to significantly affect the performance. In order to tune lambda, the validation set was used. The lambda which led to the best classifier was chosen to be the optimal one. The optimizer started at lambda=0.1, ended at lambda=10, and increased by iterations of 0.1. Figure 1 displays the performance on the training and validation sets for lambda=10. Furthermore, the optimal alpha was higher than it was in the previous project.

Finally, as seen in Figure 2, the performance on the validation set seemed to peak at 75%. This value seemed unexpectedly low, although regardless of the parameters of the algorithm the performance could never increase past this mark. It is also noted that the performance on the validation set seemed to decline slightly even though the performance on the training set seemed to continue to increase. This should be taken into consideration for the issue of overfitting; it may be preferred to end the descent after around 1000 iterations. Figure 3 displays the learning curves for MAX_ITERS=1100.

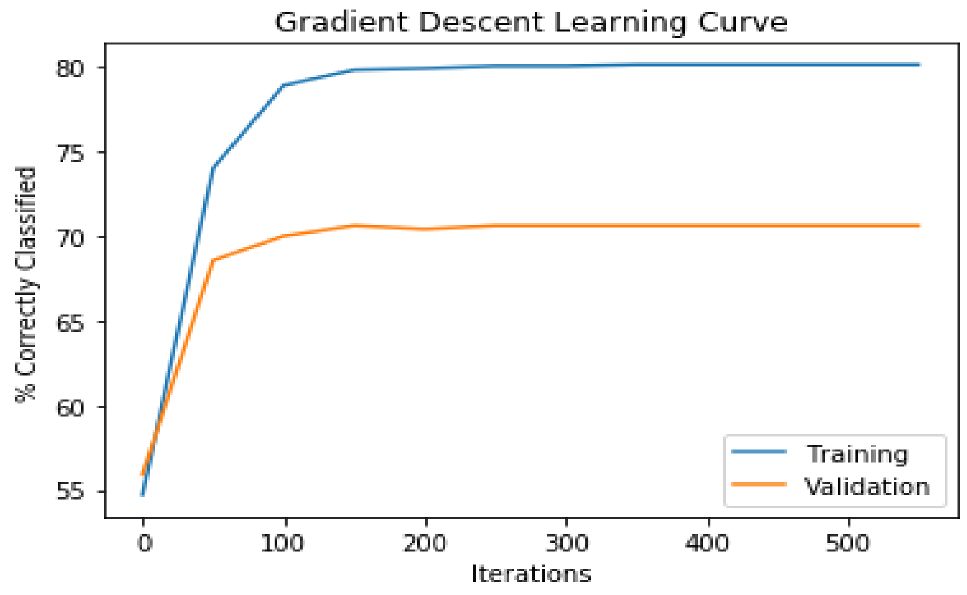

Figure 1: Learning Curves for a Logistic Regression Classifier on Training and Validation Sets. alpha=0.001, lambda=10, MAX_ITERS=10 000

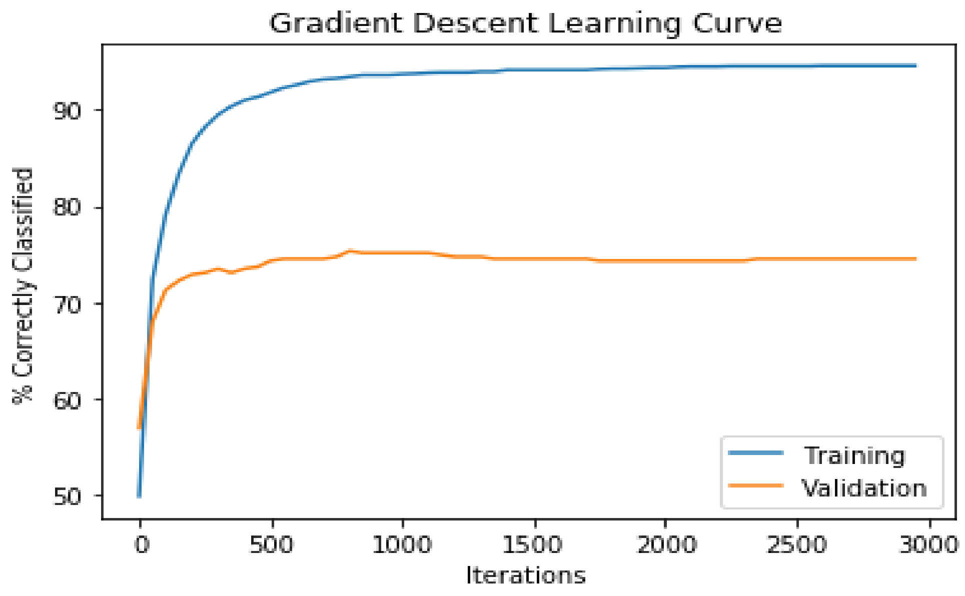

Figure 2: Learning Curves for a Logistic Regression Classifier on Training and Validation Sets. alpha=0.001, lambda=0.8, MAX_ITERS=10 000

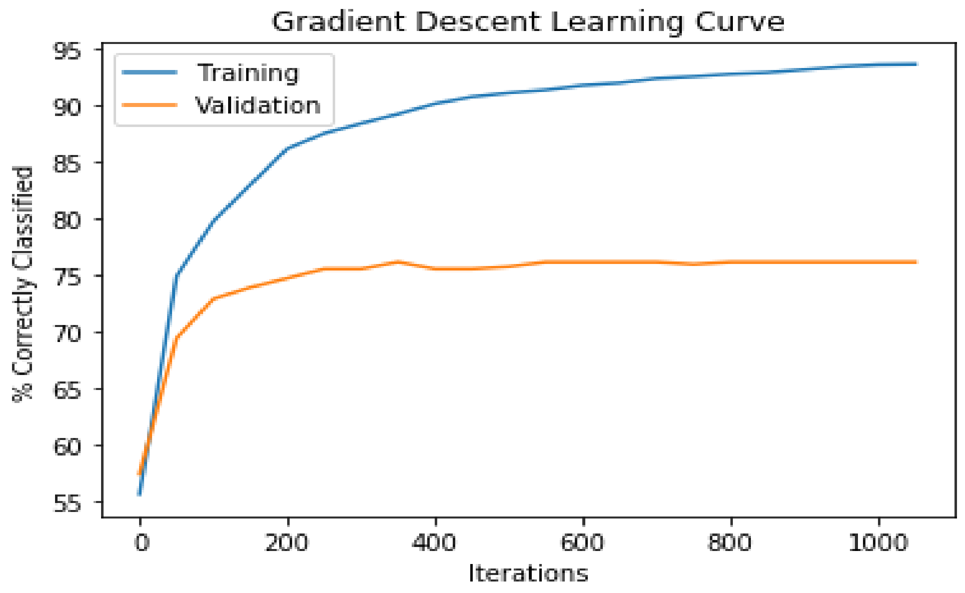

Figure 3: Learning Curves for a Logistic Regression Classifier on Training and Validation Sets. alpha=0.001, lambda=0.8, MAX_ITERS=1100

Part 5)

Logistic Regression:

For logistic regression, the generalized multicase probability of the input being of class i is

$$p_i = \frac{exp(w_0^{(i)} + w_1^{(i)}x_1 + ... + w_k^{(i)}x_k)}{\sum_j exp(w_0^{(j)} + w_1^{(j)}x_1 + ... + w_k^{(j)}x_k)} > 0.5$$

For the case of binary classification the expression can be written as

$$p_1 = \frac{exp(o_1)}{exp(o_1) + exp(o_2)} = \frac{1}{1 + exp(o_2 - o_1)} > 0.5,$$

where $o_i \equiv w_0^{(i)} + w_1^{(i)}x_1 + ... + w_k^{(i)}x_k$, and p_1 denotes the probability of the input being of class 1. Note that $p_1 = 1 - p_2$.

Thus, rearranging, we require the following inequality to hold when deciding to classify as $y = 1$:

$$1 + exp(o_2 - o_1) < 2$$
$$exp(o_2 - o_1) < 1$$

Taking the natural log of both sides:

$$o_2 - o_1 = (w_0^{(2)} + w_1^{(2)}x_1 + ... + w_k^{(2)}x_k) - (w_0^{(1)} + w_1^{(1)}x_1 + ... + w_k^{(1)}x_k) < 0$$

Grouping terms:

$$(w_0^{(2)} - w_0^{(1)}) + (w_1^{(2)} - w_1^{(1)})x_1 + ... + (w_k^{(2)} - w_k^{(1)})x_k < 0$$

Multiplying each side by -1:

$$(w_0^{(1)} - w_0^{(2)}) + (w_1^{(1)} - w_1^{(2)})x_1 + ... + (w_k^{(1)} - w_k^{(2)})x_k = \theta_0 + \theta_1 I_1(x) + ... + \theta_k I_k(x) > 0$$

By observation, it is deduced that $\theta_j = (w_j^{(1)} - w_j^{(2)})$ and $I_j(x) = x_j$. This inequality must hold for classification as $y = 1$, and must not hold for classification as $y = 0$.

Naive Bayes:
Again, the probability of headline being of class y=1 is

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

and this expression must be larger than 0.5 in order to choose to classify this headline as the class y=1. Simplifying this inequality:

$$\frac{P(headline|y = 1)P(y = 1)}{P(headline)} > 0.5$$

$$P(headline|y = 1)P(y = 1) > 0.5 \cdot P(headline)$$

Then, taking the log of both sides:

$$log(P(headline|y = 1)P(y = 1)) > log(0.5 \cdot P(headline))$$

By conditional independence:

$$log(\Pi_i P(word_i|y = 1)P(y = 1)) > log(0.5 \cdot P(headline))$$

$$log(P(y = 1)) + \sum_i log(P(word_i|y = 1)) = \theta_0 + \theta_1 I_1(x) + ... + \theta_k I_k(x) > log(0.5 \cdot P(headline))$$

By inspection, $\theta_0 = log(P(y = 1))$, $\theta_{i \neq 0} = 1$, and $I_i(x) = I_i(word_i) = log(P(word_i|y = 1))$

Part 6) *It is assumed that θ in the context of this question refers to the same θ that was derived in part 5.*

a) It is noted that θ was defined based on classifying an input as real news. The top 10 largest values of θ are:

2.21237425, 1.89837329, 1.62553059, 1.42992295, 1.39639204, 1.38342583, 1.30766623, 1.2977954, 1.27815396, 1.22090128

These correspond to the following words:

'trumps', 'korea', 'north', 'turnbull', 'trade', 'ban', 'east', 'travel', 'tax', 'australia'

The top 10 most negative values of θ are:

-2.49557114, -1.65760892, -1.56222629, -1.53497249, -1.49412982, -1.46469659, -1.45115002, -1.43763763, -1.41043447, -1.40070633

These correspond to the following words:

'hillary', 'u', 'voting', 'if', 'just', 'is', 'kill', 'wins', '4', 'voter'

The first list of words, corresponding to large values of θ, has the word 'trumps' in common with the list of words which most strongly predicts real news. However, the rest of the words are unique. It is noted that many words in the lists in part 3 (a) are stopwords; the lists in this part on the other hand contain very few.

b) In this part, stopwords have been removed. The top 10 largest values of θ are:

2.21237425, 1.89837329, 1.62553059, 1.42992295, 1.39639204, 1.38342583, 1.30766623, 1.2977954, 1.27815396, 1.22090128

These correspond to the following words:

'trumps', 'korea', 'north', 'turnbull', 'trade', 'ban', 'east', 'travel', 'tax', 'australia'

The top 10 most negative values of θ are:

-2.49557114, -1.65760892, -1.56222629, -1.49412982, -1.45115002, -1.43763763, -1.41043447, -1.40070633, -1.38123331, -1.31561228

These correspond to the following words:

'hillary', 'u', 'voting', 'just', 'kill', 'wins', '4', 'star', 'voter', 'fame'

After stopwords were removed, the list for large values of θ shared 'trumps', 'korea', 'north', and 'ban' with the list of words which most strongly predicted real news. The list of large negative values of θ shared 'hillary' and 'just' with the list of words which most strongly predicted fake news.

c) For this project, values of all inputs were binary, either being 1 if the word appears in the headline and 0 otherwise. Thus, since this normalization is present, it makes sense to use the magnitudes of the weights as indications of importance. Here, no weight could ever be multiplied by an unexpectedly large coefficient arising for example from a very large x_i relative to the other inputs. Another example of when this issue may arise is in image classification. An example was discussed in class where images of tanks were analyzed for military research; however, the input images were not normalized by brightness and thus the results were affected by how cloudy it was when the images were taken.

Part 7)

a) A decision tree was built using DecisionTreeClassifier which is included in sklearn. The trees were trained using the same inputs as those for part 4, although the bias was removed. Two parameters were tuned for these trees, specifically the *max_depth* and the *max_features* parameters, the latter of which was used with percentage values. These parameters were tuned by iterating over 10 possible values for each of them, taking on values of {5,10,25,50,100,150,300,500,750,1000} and {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0} respectively. The optimal parameters were $(max\_depth, max\_features) = (500, 0.1)$ for the training set and $(max\_depth, max\_features) = (150, 0.6)$ for the validation set. For the forthcoming parts, the optimal validation set parameters are used.

To show the relationship between *max_depth* and the performance, the value of *max_features* was held constant; this constant was chosen to be the optimal one for the validation set ($max\_features = 0.6$).

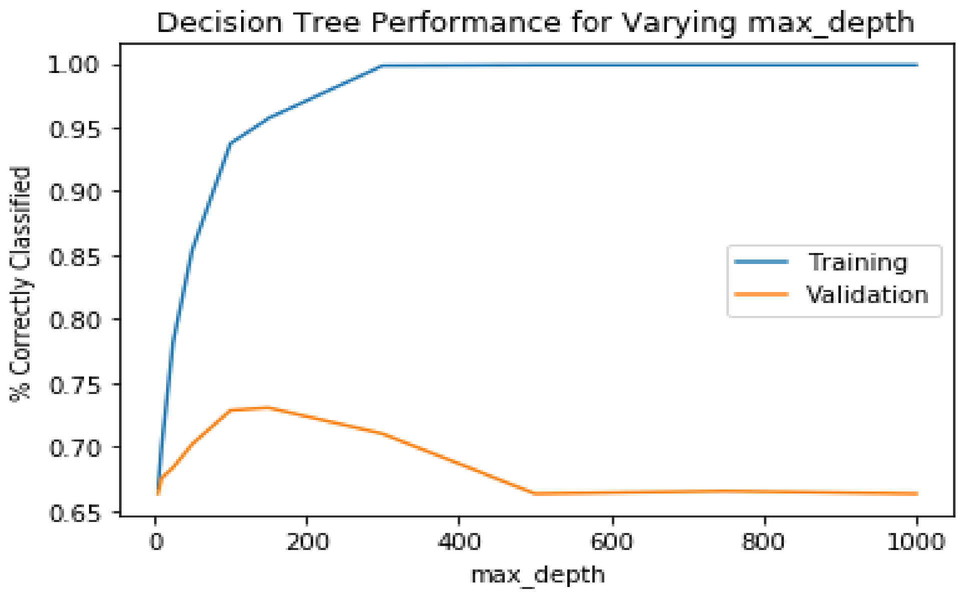

Figure 4: Percent Classification for varying max_depth sizes for max_features=0.6

b) The first two layers are shown in Figure 5, which was created by using graphviz. It is seen that the way these layers were generated were of the form $word \leq 0.5$. Since the input values were integers, this inequality represents testing whether or not the listed word is present in the headline. The True branches mean that the word is not present, and the False branches mean that the word is present. Thus, all left branches mean the word is not present, and all right branches mean the word is present.

Looking back to part 3(a), the word 'the' also appears in the list of words whose presence predict fake news. The words 'trumps' and 'on' both appear in the list of words whose presence predict real news.

Then, looking back to part 6(a), it is seen that 'hillary' appears in the list of words for most negative θ values, and the word 'trumps' appears in the list of words for largest θ values.

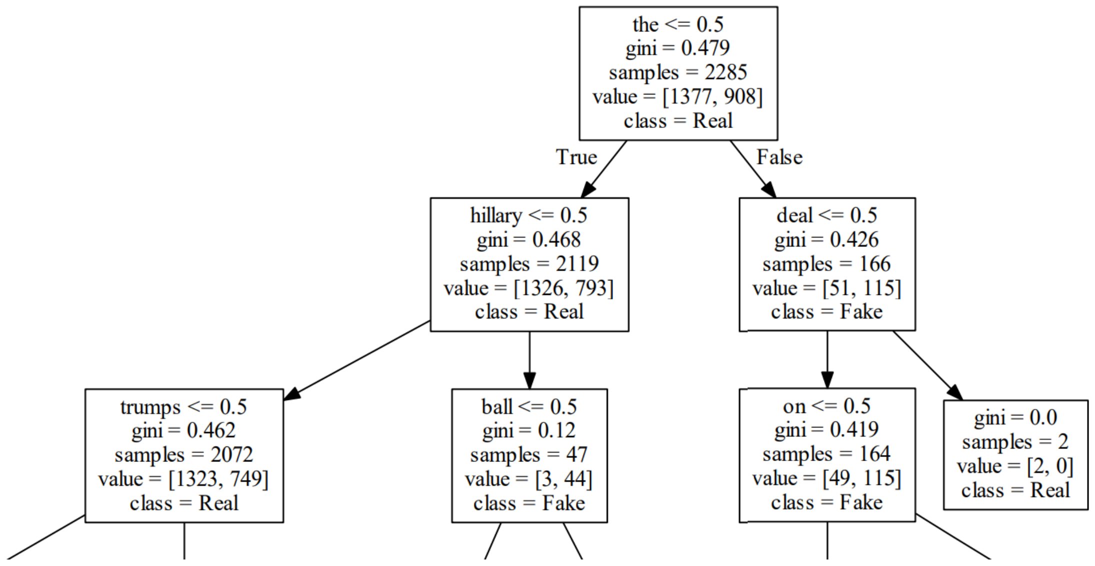

Figure 5: First two layers of the tuned decision tree

c) In the lecture slides for SVM and Kernels, overfitting was formally defined as being able to find a θ_0' such that

$$TrainPerformance(\theta_0') < TrainPerformance(\theta_0)$$

$$Performance(\theta_0') < Performance(\theta_0)$$

Thus, to determine which overfit the most, the ratio of *TrainPerformance* to *Performance (ie. the performance on all the data including the training set)* was taken; as per the above definition, more overfitting would correspond to a larger value for this ratio. It was determined that the Naive Bayes classifier overfit the most.

The classifier which performed the best was the Naive Bayes classifier, and the worst was the Decision Tree Classifier. The methodology in determining performance here was the percentage of correct classification on the test set, since this set was never seen during the training of the models.

Part 8)

a) It is first noted that the Naive Bayes assumption applies here, specifically that Y and x_i are conditionally independent. Since this was assumed for previous parts of the project, the assumption remains for consistency. Thus,

$$I(Y; x_i) = H(Y) - P(X = x_i)H(Y)$$

The code for computing this is shown in Listing 2. The resulting mutual information of the word 'the' was 0.89897.

Listing 2: Computing Entropy and Mutual Information

```python
def H(Y):
    '''
    This function calculates the entropy of random variable Y with two possible
    classes. Y is an array of length #samples, with each element being 0 or 1.
    0 corresponds to real news, 1 corresponds to fake news.
    '''
    totalcount=len(Y)
    count0=(Y == 0).sum()
    count1=(Y == 1).sum()

    p_0=count0/totalcount
    p_1=count1/totalcount

    entropy = -p_0*math.log(p_0,2) - p_1*math.log(p_1,2)
    return entropy

...

    #Compute mutual information of the top word 'the'
    wordi_index=all_words.index('the')
    xi=x_train[:,wordi_index]
    totalcount=len(y_train)

    mi_i=H(y_train)-((xi==1).sum()/totalcount)*H(y_train)
```

b) The next word that was chosen was 'trumps'. The computation for this word was exactly the same as it was for part (a), with the word 'the' changed to 'trumps'. The rationale for choosing this word was that it seemed to bear significance in the classification of headlines based on previous parts of this project. The resulting mutual information for this word was 0.94139, higher than it was for 'the'. A larger mutual information corresponds to a larger reduction in uncertainty, meaning this word had the hypothesized effect of being important to fake news classification.