CSC411 Project 3

March 19, 2018

Christopher Sheedy 1002362542, Shahin Jafari 1002401634

Contents

Part 1

The dataset consists of 1298 headlines classified as "fake" and 1968 headlines classified as "real". It is feasible to distinguish between them by using binary classifiers, and focusing on words, as features of the classifiers. To see if there are any words that may be particularly useful for classification, we took a count of every word in every headline and sorted by occurrence for both the fake and real data set. There were many common words to both sets. "trump" was the most common word for both real and fake headlines, and so is likely not useful. "donald" may be more useful. It appears 829 times in real headlines, and only 228 times in fake headlines. "us" is also a potentially useful word, appearing 230 times in real headlines, and only 39 times in fake headlines. It might be helpful to know if "us" is a abbreviation for "United States" or if it is referring to the pronoun. "says" may also be helpful, appearing 178 times in real headlines, and only 47 times in fake headlines. We tried to look at words that were unique to either set of headlines but this was not instructive, since unique words were usually one-offs, and only appeared once or twice in either set.

Part 2

To determine whether or not a headline was fake news we trained a naive bayes classifier. With this calculator we calculated the probability of a set of features given a label of "fake" or "real" based on a training set. A possible problem with this is in the calculation:

$$p(x_1, x_2, ..., x_n|c) = p(c) \prod_{i=1}^{n} p(c|x_i)$$

since the multiplication of many small numbers can lead to arithmetic underflow. To solve this problem we took advantage of the fact that

$$\prod_{i=1}^{n} p(c|x_i) = \exp\left(\sum_{i=1}^{n} \log p(c|x_i)\right)$$

and so turned the product into a sum. In this way, we could take the sum of the log likelihoods and exponentiate it to recover the original value, thus avoiding underflow.

$p(c)$ was calculated as $\frac{count(c)}{count(c)+count(c')}$. $p(c|x_i)$ was calculated as $\frac{count(x_i)+m*p}{count(c)+m}$. The parameters m and p were tuned using a brute force method. the parameters were varied over a large range to find a suitable an optimal solution. For m we experimented with values between 0 and 100, and for p we tried values between 0.001 and 0.1. With $m = 55$ and $p = 0.004$ we got an accuracy of 79 percent, so these parameters were used. On the training set this gave us an accuracy of 93.17 percent, and on the test set we had an accuracy of 77.30 percent.

Part 3

Part 3 a)

To determine the influence of words we decided to look at $p(fake|word)$, $p(fake|\neg word)$, $p(real|word)$, $p(real|\neg word)$. $p(fake|word)$ and $p(real|word)$ can be calculated directly, or using bayes rule:

$$p(c|w) = \frac{p(w|c)p(c)}{p(w)} = \frac{\frac{count(w,c)}{count(c)}\frac{count(c)}{count(c+\neg c)}}{\frac{count(w)}{count(c+\neg c)}} = \frac{count(w,c)}{count(w)}$$

To calculate $p(fake|\neg word)$ and $p(real|\neg word)$ we used bayes rule again:

$$p(c|\neg w) = \frac{p(\neg w|c)p(c)}{p(\neg w)} = \frac{count(\neg w,c)}{count(\neg w)}$$

Using this method, we got the following results

Table 1: top ten predictors for news

Class	words
"real"	'korea', 'turnbull', 'travel', 'australia', 'paris', 'tax', 'comments', 'trumps', 'refugee', 'congress'
"fake	'trump', 'the', 'to', 'hillary', 'a', 'for', 'of', 'in', 'and', 'is'

Table 2: top ten predictors for news (by absence)

Class	words
"real"	'comment', 'watch', 'black', 'just', 'm', 'cnn', 'won', 'you', 'behind', 'voter'
"fake	'ban', 'north', 'climate', 'trade', 'immigration', 'james', 'malcolm', 'leaders', 'decision', 'ahead'

Presence has a much greater affect on the classification of the headline than absence. Consider what it means for $p(class|word)$ to be high. Using our approximation for $p(class|word)$, this means that word has appeared many times in headlines of a particular class. Generally, we have found this to be a good method for classifying. Consider now what it means for $p(class|\neg word)$ to be high. Using our approximation, this means that there are very many headlines of a particular class that do not contain the word. So, words with high $p(class|\neg word)$ are ones that only appear once, or never in that particular class. However, this is not very useful for classification. Many of these words are one-offs that do not represent anything useful for classification. Furthermore, we are very likely to get ties. The number of one-offs or words that appear twice is typically much higher than say, the number of words that appear 400 times, or 401 times. So for these reasons, the presence of words is much more important than the absence.

Part 3 b)

For this section we have removed all english stop words from the tables for $p(x_i|"real")$ and $p(x_i|"fake")$. With those words gone the words whose presence likely indicates real/fake are:

Table 3: top ten predictors for news

Class	words
"real"	'korea', 'turnbull', 'travel', 'australia', 'paris', 'tax', 'comments', 'trumps', 'refugee', 'congress'
"fake	'trump', 'hillary', 'just', 'donald', 'clinton', 'comment', 'watch', 'new', 'win', 'breaking'

Table 4: top ten predictors for news (by absence)

Class	words
"real"	'comment', 'watch', 'black', 'just', 'm', 'cnn', 'won', 'voter', 'supporter', 'voting'
"fake	'ban', 'north', 'climate', 'trade', 'immigration', 'james', 'malcolm', 'leaders', 'decision', 'ahead'

Part 3 c)

It might make sense to keep stop words if you believed that the presence or lack of stop words in a headline were indicative of it being fake news or not. If there was some relationship between fake news and the use of stop words then keeping them would make your classifier more accurate. This might in fact be the case for our dataset. Of the top ten words indicating real/fake news, 0 of the real news indicators are stop words, while 8 of the fake news indicators are stop words.

However, if you believe that stop words are uniformly distributed among both real and fake headlines, then including them increases the risk of overfitting to the random noise in the training set. For example, you could have a stop word that appears several times in the fake headlines and never in the real headlines simply by chance. In this case your classifier will assign a relationship that is not really there, and thus reduce your accuracy.

Part 4

In order to build a logistic regression classifier, we first had to build the vocabulary, the column vector of all the words that appeared in all the headlines. A dedicated function for this purpose was then made. The reason that we are using all the keywords in all of the data is that in a real world, we would use an actual dictionary to build our vocabulary, just to make sure that we are accounting for all the possible words that can potentially be in a headline. This is however very costly to do for this project, but using all the headlines has the similar rationale.

After building the vocabulary, the training set, validation set and test set were made to adhere the same form as the vocabulary. Each training/validation/test example is represented using a column matrix, where the rows are the words found in the vocabulary.

Then, the same procedure as linear regression was applied to training set, with the difference that the hypothesis function was represented as:

$$h(\theta) = \sigma(\theta^T x)$$

and the cross entropy loss was used instead of a square difference loss to maximize the log likelihood of the data. Since the number of features were enormous (about 5000 thousands) and the number of samples for the training set were also very large, the training phase was costly in terms of computation time. As a result, we used stochastic gradient descent with a batch size of 100, randomly chosen from the training matrix.

After building this part, we ran an exhausting search to find the optimum parameters for the learning rate, maximum iterations, error bound and regularization parameter for cross validation. Note that L2 regularization was used for this part to avoid deriving coefficients to zero. The table of the results can be found in appendix. From the results, we decided to use the learning rate of 0.001, maximum iterations of 10000, error bound of 0.00001 and λ of 1. Then the classifier was tested against the test set, yielding 79.35% accuracy and the learning curve was plotted. Below is the learning curve:

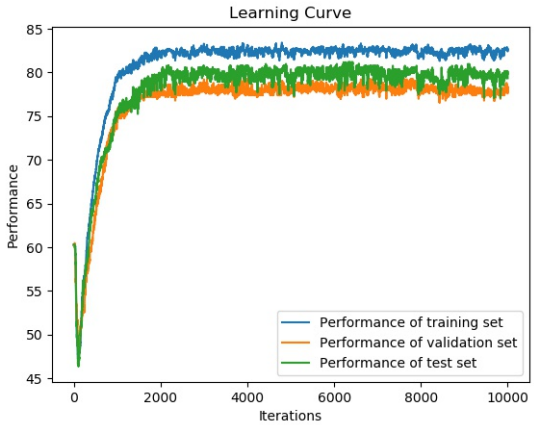

Figure 1: Learning Curve. Performance vs. Number of Iterations

Part 5

In the naive bayes model, Is are indicator variables of a keyword appearing in a headline. θ are the probabilities of that word, approximated using the number of appearance of the word in a particular class. We can also specifically describe θ_0 as the learned probability of words in the headline that are not present in the training set, using parameters m and p.

In logistic regression model, Is are indicator variables of a keyword appearing in a headline and θs are weights or coefficients associated with each keyword as in the linear regression model. θ_0 is the bias term which we incorporate in the θ column vector, and for it to generate the sum, we added a row of 1 to the X matrix. Overall, $\sigma(\theta^T X)$ gives us the probability of a sample headline being real, or $P(real|\theta, X)$. Of course, if this probability was greater or equal to 0.5, the classifier would predict it as a real headline.

Part 6

Part 6a)

Below are the top positive and negative weights and their corresponding keyword from the logistic regression model with learning rate of 0.001, maximum iterations of 10000, error bound of 0.00001 and λ of 1, including stopwords.

Table 5: Top Positive θs

Weight	Word
0.771420745	trump
0.656843384	new
0.606545342	donald
0.549694387	election
0.364528264	media
0.233048857	executive
0.228370413	comments
0.223067601	calls
0.213389331	korea
0.177543478	gets

Table 6: Top Negative θs

Weight	Word
-0.251081643	remember
-0.25509692	court
-0.256198719	masks
-0.256842566	concedes
-0.27172933	open
-0.300897036	trumps
-0.330844409	beginning
-0.402406323	load
-0.567537486	clinton
-0.70630106	year

In logistic regression of this project, top positive are indicators of a real news and top negatives are indicators are fake news. In comparison with 3a), there are 2 similar words that their presence indicates real news (comments,korea) and there are no similar words that are the same for their presence to indicate a fake news. In terms of keywords that their absence indicates real news, we have no matching between the keywords in logistic regression and naive bayes. It is also worth noting that most of the keywords in part 3a for fake news are stopwords.

Part 6 b)

Note that in our table for part 6 a, we did not have any stop words with a high weight. As a result, the same table applies to this part as well:

Table 7: Top Positive θs

Weight	Word
0.771420745	trump
0.656843384	new
0.606545342	donald
0.549694387	election
0.364528264	media
0.233048857	executive
0.228370413	comments
0.223067601	calls
0.213389331	korea
0.177543478	gets

Table 8: Top Negative θs

Weight	Word
-0.251081643	remember
-0.25509692	court
-0.256198719	masks
-0.256842566	concedes
-0.27172933	open
-0.300897036	trumps
-0.330844409	beginning
-0.402406323	load
-0.567537486	clinton
-0.70630106	year

The reason for not having any stopwords with high weight, could be that they frequently appear in both real and fake headlines, meaning that their presence does not add any value/evidence for a headline being real or fake in logistic regression, since in this model we are maximizing the log-likelihood of the data.

In comparison with 3b), there are 2 similar words for real news (korea, comments) and there is only 1 word (clinton) that is similar for fake news. There are also some words that are probably generated in a headline together, but each model assigned different weights for them. For example, north and korea are most likely to be generated together, but we only have korea in logisitc regression model.

Part 6 c)

Using the magnitude of the logistic regression parameters is usually not a good measure of importance, since it can be very dependent on the scale of the features. Consider for example a situation where we are building a classifier for the price of homes. The features could be square footage, number of bedrooms, and number of bathrooms. The square footage will likely be on the scale of 10^3 while the number of bedrooms/bathrooms would likely be on the scale of 10^0. Thus the feature vector will be dominated by the square footage term. now, due to the relative difference between these scales, the parameter for square footage will likely be a lot smaller than the parameters for bedroom/bathroom, even if it were more important for pricing than bedroom/bathroom. Thus, we can see that in some cases, the scale of the problem can affect the parameter size more than the importance of that feature. Now for our particular problem, the features are all either 0, or 1, so the problem of scale does not affect us. Additionally, headlines generally have a similar number of words, and so the magnitude of the feature vectors is relatively constant for all training examples, and normalization would likely not have a great effect. For these reasons, it is not an especially bad idea to use magnitude of parameters to rank importance of the features.

Part 7

Part 7a)

In this part, we used the same training/validation/test set as the previous parts and we used sklearn.tree to build a decision tree classifier. In order to find out the relationship between the depth and the performance on the training/validation set, we tried different depth, starting from 0 to 1000, with increments of 5. Since depth of 0 did not have any real meaning, we changed it with the depth of 1. Below is the result:

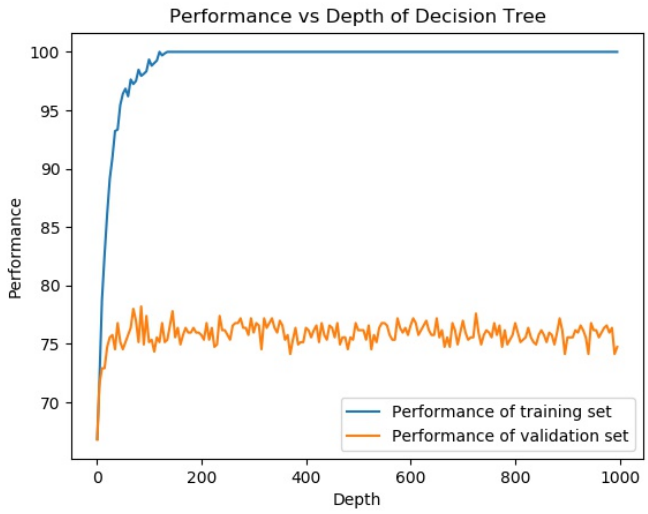

Figure 2: Decision Tree Visualization

The highest performance of the decision tree was when the depth was set to 85. As a result, the maximum depth was set to be 85 and the classifier was tested against the test set, yielding 74.4376%. All the other parameters of the decision tree was set to default value.

Part 7 b)

To generate a PDF file of the decision tree containing pictures, we installed graphviz library. This is also a requirement to reproduce the same graph if needed. Below is the visualization of the first two layers:

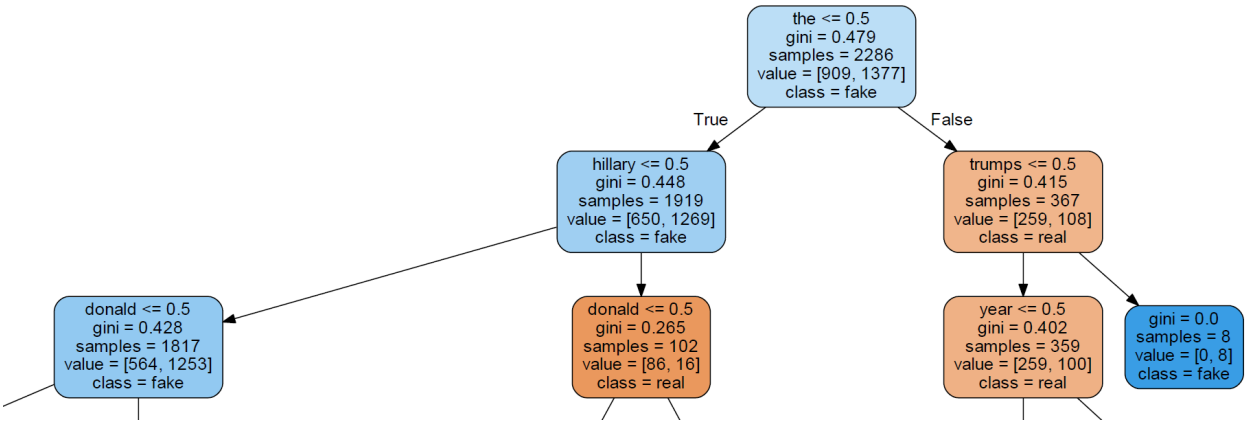

Figure 3: Decision Tree Visualization

As can be seen from the decision tree, the keywords 'the' and 'hillary' are again taken as a strong evidence for a news being fake and the keywords 'trumps' and 'donald' are taken as strong evidence for a news being real which is consistent with previous models.

Part 7 c)

Below are the results of different models used in this project:

Table 9: Summary of performances

	Training Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Difference Between Training and Validation
Naïve Bayes	93.1	79.8	77.3	13.3
Logistic Regression	82.59	78	79.35	4.59
Decision Tree	97.94	75.97	74.44	21.97

From the table above, one can notice that the the logistic regression has the best performance, and the decision tree has the worst performance on the test set. From the difference between the training set performance and the validation set performance, we can also realize that the decision tree model overfits the training set the most, since the performances on the training set is very good but on the test set, it does not perform well. Logistic Regression does better than naive bayes, simply due to the fact that it does not make the independence assumption regarding the keywords. Their performances are similar, but only because the headlines are short. Had the headlines were longer with more keywords, we would have expected that the difference between the logistic regression and the naive bayes become larger.

Part 8

Mutual information is computed as:

$$I(Y;X) = H(Y) - H(Y|X) = H(Y) - \Sigma_x P(X=x)H(Y|X=x)$$

where

$$H(Y) = -P(Y)log_2P(Y)$$

For each split, we can calculate H(Y) by using the number of samples in each class at time of using the keyword to split.

Part 8 a)

$I(Y;X)$ in this case becomes:

$$H(unsplitted) - (P(has_{keyword})H(splitted|has_{keyword}) + P(no_{keyword})H(splitted|no_{keyword}))$$

To calculate $I(Y;X)$, a function was written to calculate H. Another function was also written to calculate $I(Y;X)$.

```
def H(split_l, split_r):
    total = float(split_l + split_r)
    return −1∗split_l/total∗np.log2(split_l/total) − split_r/total∗np.log2(split_r/total)

def part_8(top,left,right):
    children = [left,right]
    summation = 0
    for child in children:
        summation += float(sum(child))/sum(top) ∗ H(child[0],child[1])
    return H(top[0],top[1]) − summation

def part_8_a():
    ###data from visualization of first layer
    top_node_split = (909,1377)
    left_child_split = (650,1269)
    right_child_split = (259,108)
    print part_8(top_node_split,left_child_split,right_child_split)
```

The code above, calculated the mutual information of the first split to be 0.0539.

Part 8 b)

In this part, the mutual information of the following layer was computed, at the node where we divide based on the keyword 'donald':

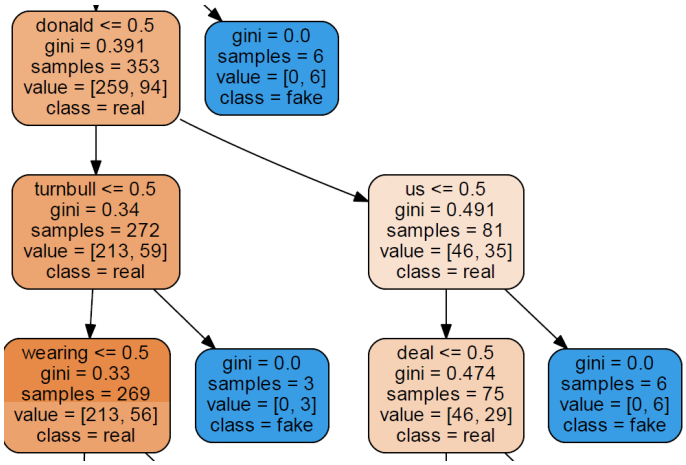

Figure 4: A layer of the decision tree

```
def H(split_l, split_r):
    total = float(split_l + split_r)
    return −1∗split_l/total∗np.log2(split_l/total) − split_r/total∗np.log2(split_r/total)

def part_8(top,left,right):
    children = [left,right]
    summation = 0
    for child in children:
        summation += float(sum(child))/sum(top) ∗ H(child[0],child[1])
    return H(top[0],top[1]) − summation

def part_8_b():
    ###data from visualization
    top_node_split = (259,94)
    left_child_split = (213,59)
    right_child_split = (46,35)
    print part_8(top_node_split,left_child_split,right_child_split)
```

The computed $I(Y;X)$ for this layer is 0.0283.

The mutual information in part a was 1.9× higher than the mutual information in part b. Mutual information can be thought as a reduction in uncertainty about a news being real or fake in this project. As a result, it makes sense for the part a number to be larger, since in the very first layer of our decision tree, we want to gain as much as information as possible, to reduce our

decision tree height. As a consequence, deeper nodes have a smaller mutual information, since had it been otherwise, we would have used that keyword higher in the tree.

Appendix

Table 10: Top results of searching for optimum parameters, part 4

learning rate	maximum iterations	error	lambda	validation set performance	train set performance
0.01	50000	0.001	1	79.42973523	83.37707787
0.01	50000	0.0001	1	79.42973523	83.37707787
0.01	50000	1.00E-05	1	79.42973523	83.37707787
0.01	50000	1.00E-06	1	79.42973523	83.37707787
0.01	1000	0.001	1	78.4114053	83.02712161
0.01	1000	0.0001	1	78.4114053	83.02712161
0.01	1000	1.00E-05	1	78.4114053	83.02712161
0.01	1000	1.00E-06	1	78.4114053	83.02712161
0.0001	50000	0.0001	1	78.20773931	82.72090989
0.0001	50000	1.00E-05	1	78.20773931	82.72090989
0.0001	50000	1.00E-06	1	78.20773931	82.72090989
0.001	10000	0.001	1	78.00407332	82.58967629
0.001	10000	0.0001	1	78.00407332	82.58967629
0.001	10000	1.00E-05	1	78.00407332	82.58967629
0.001	10000	1.00E-06	1	78.00407332	82.58967629
0.001	50000	0.001	1	77.80040733	82.72090989
0.001	50000	0.0001	1	77.80040733	82.72090989
0.001	50000	1.00E-05	1	77.80040733	82.72090989
0.001	50000	1.00E-06	1	77.80040733	82.72090989
0.01	10000	0.001	1	77.39307536	81.3648294
0.01	10000	0.0001	1	77.39307536	81.3648294
0.01	10000	1.00E-05	1	77.39307536	81.3648294
0.01	10000	1.00E-06	1	77.39307536	81.3648294
0.0001	50000	0.001	1	77.39307536	81.75853018
0.01	1000	0.001	2	76.98574338	80.70866142
0.01	1000	0.0001	2	76.98574338	80.70866142
0.01	1000	1.00E-05	2	76.98574338	80.70866142
0.01	1000	1.00E-06	2	76.98574338	80.70866142
0.001	1000	0.001	2	76.98574338	80.1399825
0.001	1000	0.0001	2	76.98574338	80.1399825
0.001	1000	1.00E-05	2	76.98574338	80.1399825
0.001	1000	1.00E-06	2	76.98574338	80.1399825
0.001	10000	0.001	2	76.17107943	79.65879265
0.001	10000	0.0001	2	76.17107943	79.65879265
0.001	10000	1.00E-05	2	76.17107943	79.65879265
0.001	10000	1.00E-06	2	76.17107943	79.65879265
0.001	1000	0.001	3	75.96741344	78.0839895
0.001	1000	0.0001	3	75.96741344	78.0839895
0.001	1000	1.00E-05	3	75.96741344	78.0839895
0.001	1000	1.00E-06	3	75.96741344	78.0839895
0.0001	10000	0.001	2	75.76374745	79.39632546
0.0001	10000	0.0001	2	75.76374745	79.39632546
0.0001	10000	1.00E-05	2	75.76374745	79.39632546
0.0001	10000	1.00E-06	2	75.76374745	79.39632546
0.01	100	0.001	2	74.9490835	78.56517935