CSC411: Project 3

Due on Monday, March 19, 2018

Robert Bazzocchi and Sean Doughty

March 20, 2018

Part 1

Describing the Datasets

There are two datasets provided for this project - a set of real news headlines and set of fake news headlines. The real news dataset and the fake news dataset consist of 1968 and 1298 headlines, respectively. Thus, the real news dataset contains nearly 700 more headlines than that of the fake news.

It is difficult to predict the credibility of these news headlines simply by observation. However, there may be some keywords that indicate with a higher probability that a headline is fake or real.

For example, take the words, "reason", "hillary", and "new". These keyword strings were counted in both the real news dataset and the fake news dataset, however, the counts vary drastically. Thus, seeing one of these words in a news headlines may cause us to believe it is within the category that has the larger count.

These counts are summarized in the table below:

Keywords	**Real Headlines Count**	**Fake Headlines Count**
reason	0	18
hillary	24	150
new	69	114

Table 1: Counts of Keywords that may be useful in Headline Classification

Part 2

Naive Bayes Implementation

In this part of the project, the Naive Bayes model was implemented. To do so, a fundamental assumption was made - the probabilities of each of the words in the set being present in a headline are *statistically independent* given a class. That is,

$$p(w_1, w_2, ..., w_n|y = c) = \prod_{i=1}^{n} p(w_i|y = c) \tag{1}$$

where $w_1, w_2, ..., w_n$ are the n words in a headline. This assumption is then used with Bayes' Theorem to rearrange the conditional probabilities, $p(y = c|w_1, w_2, ..., w_n)$ into probabilities we know (i.e., $p(w_i|y = c)$).

The code and function calls for this part of the project are summarized as follows:

1 *get_datasets()* obtains the training, validation, and test sets

2 *create_count_dict(x_train,y_train)* creates a dictionary where the keys are "real" or "fake" and the values are subdictionaries with keys being words and values being the number of headlines that that word occurs in given the class.

3 *get_optimal_parameters(x_train, y_train, x_val, y_val)* finds and returns the optimal m and p Bayes' model parameters (discussed in more detail below)

4 *create_prob_dist(x_train, y_train)* creates a dictionary where the keys are "real" or "fake" and the values are subdictionaries with keys being words and values being their conditional probabilities $p(w_i|y = c)$

5 *test_naive_bayes(P,x_val,y_val)* returns the accuracy of the Bayes' model given the predictions from conditional probability dictionary P and the actual classifications of the headlines, y_val

Tuning the Bayes' Model Parameters

To find the optimal values for parameters m and p, a sweep was performed through a range of values, with *test_naive_bayes(P,x_val,y_val)* called in every iteration. The values swept through for m were all multiples of 10 in range 10 to 1010, and the values swept through for p were nine values in range 0.00005 to 0.50000. This function checks if the accuracy at each iteration is larger than the previous greatest accuracy. If so, it saves the parameters in a variable called *best_parameters* and returns it after all values have been swept through. For reference, this function is included on the following page.

Code for Obtaining Optimized Bayes' Model Parameters

```
def get_optimal_parameters(x_train,y_train,x_val, y_val):
     """
     FUNCTION:
          This function takes as input the training dataset pair (x_train, y_train)
          and the validation dataset pair (x_val, y_val). The former is used to build
          the conditional probability distribution dictionary and the latter is used
          to test the naive bayes' algorithm with varying parameter values. This
          function sweeps through the a wide range of m and p combinations, and returns
          the parameters m and p that yield the highest accuracy on the validation set.
     INPUT:
          x_train (list) : a list of headlines containing ~70% of the total data
          y_train (list) : a list of classifications corresponding to x_train headlines
          x_val (list)   : a list of headlines containing ~15% of the total data
          y_val (list)   : a list of classifications corresponding to x_val headlines
     OUTPUT:
          m (int)             : naive bayes' parameter
          p (float)      : naive bayes' paremeter
     """
     best_accuracy = 0
     best_parameters = [None,None]
     for m in range(10,1010,10):
          for p in [0.00005,0.0001,0.0005,0.001,0.005,0.01,0.05,0.10,0.50]:
               P = create_prob_dist(x_train,y_train,m,p)
               accuracy = test_naive_bayes(P,x_val,y_val)
               if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_parameters = [m,p]
     m = best_parameters[0]
     p = best_parameters[1]
     return m, p
```

Results from Obtaining Optimized Bayes' Model Parameters

The optimal parameters of m = 970 and p = 0.0005 yielded an accuracy of 81.557% on the test set.

Part 3

Part 3a: Top 10 Words with Most Influence in Classification

In order to determine which words were the most influential in classification when present, the *P_classes* dictionary was used. This contains all the conditional probabilities, $p(c|w)$, which in English, is the probability of the headline being of class c given that word w exists within it. Thus, iterating through each word in this dictionary and finding the largest 10 probabilities would yield the most influential words when present.

Similarly, this was done with the dictionary, *P_classes_negated*, which consists of all the conditional probabilities, $p(c|notw)$. This dictionary was built in a similar manner to that of *P_classes*, with $p(w|c)$ replaced with $1 - p(w|c)$.

Part 3a: Code (without documentation)

```python
def get_top10s(P_classes, P_classes_negated,no_stopwords=False):
    # 10 WORDS WHOSE PRESENCE MOST STRONGLY PREDICT THE NEWS IS REAL AND FAKE
    top_10_present = {"real": {}, "fake": {}}
    for classification in top_10_present:
        for word in P_classes:
            if no_stopwords and word in ENGLISH_STOP_WORDS: continue
            if len(top_10_present[classification].keys()) < 10:
                top_10_present[classification][word] = P_classes[word][
                    classification]
                continue
            if classification in P_classes[word]:
                if P_classes[word][classification] > min(top_10_present[
                    classification].values()):
                     word_of_min_prob = min(top_10_present[classification], key=
                         top_10_present[classification].get)
                     top_10_present[classification].pop(word_of_min_prob)
                     top_10_present[classification][word] = P_classes[word][
                         classification]

    # 10 WORDS WHOSE ABSENCE MOST STRONGLY PREDICT THE NEWS IS REAL AND FAKE
    top_10_absent = {"real": {}, "fake": {}}
    for classification in top_10_absent:
        for word in P_classes_negated:
            if no_stopwords and word in ENGLISH_STOP_WORDS: continue
            if len(top_10_absent[classification].keys()) < 10:
                top_10_absent[classification][word] = P_classes_negated[word][
                    classification]
                continue
            if classification in P_classes_negated[word]:
                if P_classes_negated[word][classification] > min(top_10_absent[
                    classification].values()):
                     word_of_min_prob = min(top_10_absent[classification], key=
                         top_10_absent[classification].get)
                     top_10_absent[classification].pop(word_of_min_prob)
                     top_10_absent[classification][word] = P_classes_negated[word
                         ][classification]

    return top_10_present, top_10_absent
```

Part 3a: Results (including stopwords)

The following tables summarize the results when running *get_top10s(P_classes, P_classes_negated,no_stopwords)* with *no_stopwords = False*.

Top 10 Words	Probabilities	Top 10 Words	Probabilities
refugee	0.9978943794662314	u	0.9968527401444383
ban	0.9994440072129851	reporter	0.9951200146818246
travel	0.9992274001673978	3	0.9968527401444383
paris	0.9980254137798387	7	0.9951200146818246
korea	0.99939061744842	breaking	0.9974054786722873
australia	0.9988278174077717	soros	0.9966121574073665
asia	0.9973709938352271	woman	0.9960007338588345
turnbull	0.9991665163645754	steal	0.9951200146818246
tpp	0.9973709938352271	secret	0.9951200146818246
climate	0.9986245713370274	m	0.9963317490886994

Table 2: Top 10 Influential Words in Classifying Real News (left) and Fake News (right) when Present

Top 10 Words	Probabilities	Top 10 Words	Probabilities
the	0.6280262546033686	donald	0.45065350208962035
is	0.6115800308702755	trump	0.4145409365826473
to	0.6093099685090617	says	0.4076426939332723
of	0.6106715354556946	trumps	0.4145308473284344
a	0.614935998746185	us	0.4115790023724735
and	0.6120419032555473	ban	0.4039828861288244
for	0.6093768507409046	travel	0.4021311708238331
hillary	0.616576240033799	north	0.4036924660155314
clinton	0.6099391775542278	korea	0.4034023958213317
just	0.6096509878293846	turnbull	0.4017858624009939

Table 3: Top 10 Influential Words in Classifying Real News (left) and Fake News (right) when Absent

Part 3b: Results (excluding stopwords)

The following tables summarize the results when running *get_top10s(P_classes, P_classes_negated,no_stopwords)* with *no_stopwords = True.*

Top 10 Words	**Probabilities**	**Top 10 Words**	**Probabilities**
refugee	0.9978943794662314	u	0.9968527401444383
ban	0.9994440072129851	reporter	0.9951200146818246
travel	0.9992274001673978	3	0.9968527401444383
paris	0.9980254137798387	7	0.9951200146818246
korea	0.99939061744842	breaking	0.9974054786722873
australia	0.9988278174077717	soros	0.9966121574073665
asia	0.9973709938352271	woman	0.9960007338588345
turnbull	0.9991665163645754	steal	0.9951200146818246
tpp	0.9973709938352271	secret	0.9951200146818246
climate	0.9986245713370274	m	0.9963317490886994

Table 4: Top 10 Influential Words in Classifying Real News (left) and Fake News (right) when Present

Top 10 Words	**Probabilities**	**Top 10 Words**	**Probabilities**
america	0.6057977422879913	donald	0.45065350208962035
new	0.6059460075766396	trump	0.4145409365826473
win	0.6066807962893278	says	0.4076426939332723
voter	0.6057973888763243	trumps	0.4145308473284344
just	0.6096509878293846	australia	0.4005247892929582
victory	0.605910729806768	ban	0.4039828861288244
watch	0.6062019227690136	travel	0.4021311708238331
hillary	0.616576240033799	north	0.4036924660155314
clinton	0.6099391775542278	korea	0.4034023958213317
black	0.6060557690168401	turnbull	0.4017858624009939

Table 5: Top 10 Influential Words in Classifying Real News (left) and Fake News (right) when Absent

Part 3c: Significance of Removing Stopwords

The key to understanding when removing stop words is beneficial to your model is knowing what is more telling of ”fake” news when given a dataset - the *context* of the headlines or the *writing style* of the headlines.

If the former is true and context is the key to differentiating ”fake” headlines from ”real” ones, then removing stopwords would be a good idea. This prevents common words that do not carry much contextual value (i.e., ”is”, ”the”, ”just”) from having large influence on classification (as seen in *Part 3b: Results (including stopwords)*).

If the latter is true, however, and the writing style is more important in differentiating headlines, then it may be good to keep stopwords. This is due to the fact that some stopwords can be made into slang (i.e., ”you” to ”u”, or ”something” to ”smth”). If these stopwords are less likely to exist in the headline, then the writing style may be assumed to be less formal, and as such the headline would have a higher probability of being classified as ”fake” news.

Part 4

To train the model, the following setup was used in PyTorch to use Logistic Regression to predict if the headline was real or fake.

- Learning rate = 0.5
- Loss function: Cross Entropy
- Optimizer: Stochastic Gradient Descent
- Activation function: Sigmoid
- Layers: 1 fully connected
- Regularization: None
- Mini-batches: None; ran full data set every time
- Iterations: 1000

Below are some code samples showing the setup of the optimization.

```
class Model(torch.nn.Module):

    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(5833, 1)  #5832 words + a bias

    def forward(self, x):
        """
        Using the sigmoid function
        """
        y_pred = F.sigmoid(self.linear(x))
        return y_pred
```

```
model = Model()

#using the cross entropy loss function
criterion = torch.nn.BCELoss(size_average=True)
#using Stocastic Gradient Descent
optimizer = torch.optim.SGD(model.parameters(), lr=0.5, weight_decay=0)

#Seeting up arrays to record the learning rates
iterations = 1000
x = np.linspace(0,iterations, iterations)
train_track = np.zeros(iterations,)
val_track = np.zeros(iterations,)

# Training loop
for epoch in range(iterations):
     y_pred = model(x_train)
     # Compute and print loss
     loss = criterion(y_pred, y_train)

     # Zero gradients, perform a backward pass, and update the weights.
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

After training the following Accuracy was obtained. Note that early stopping was used as the success of the validation set was starting to decrease.

- Training set: 91.1%
- Validation set: 82.4%
- Test set: 86.3%

A lot of experimentation was conducted to modify the weight decay parameter to improve the results. With both SGD and ADAM, no improvement could be observed by adding regularization. In fact, with high weight decay values the accuracy would begin to decrease. The final learning rates can be visualized in Figure 1.

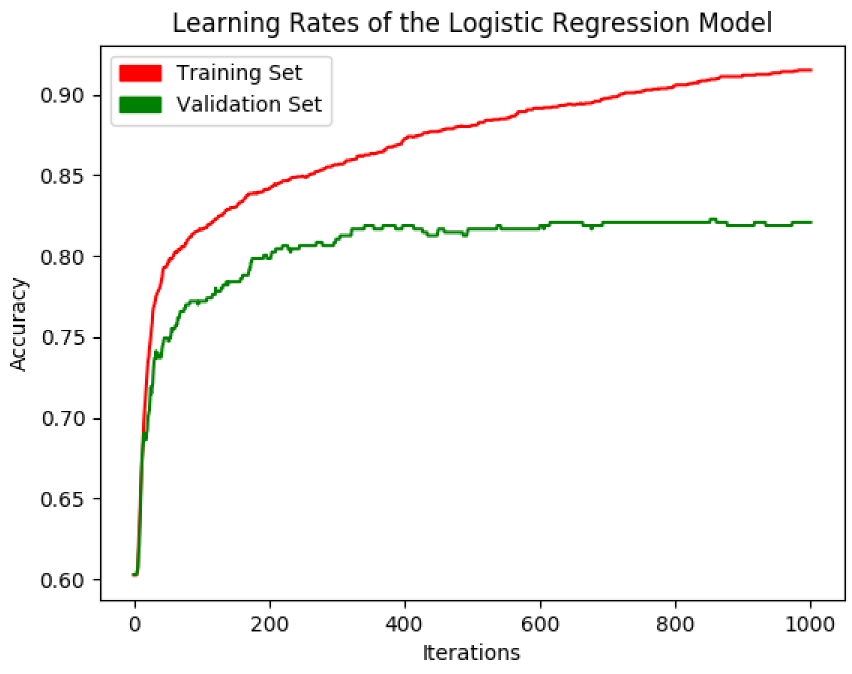

Figure 1: Learning Rates

Part 5

Given the equation:

$$\theta_0 + \theta_1 I_1(x) + \theta_2 I_2(x) + ... + \theta_k I_k(x) > thr \tag{2}$$

The terms mean the following when it comes to test:

Logistic Regression

- $I_n(x)$ is 1 if the n$^{\text{th}}$ word is in the test headline, or 0 otherwise.
- θ_n is the trained weight from the training.
- θ_0 specifically refers to the value of the bias.

Naive Bayes

- $I_n(x)$ is 1 if the n$^{\text{th}}$ word in the headline is a word to consider in classification, or 0 otherwise.
- θ_n is the log of the conditional probabilities, $\log(p(w_i|c))$.
- θ_0 specifically refers to the the log of the prior probabilities, $\log(p(c))$.

Part 6

Part 6a: Highest Weights from Logistic Regression (including stopwords)

These words are most similar to the words in 3a whose absence most strongly predicts that the news is real or fake. Real being 1 and fake being 0.
Of the 10 highest positive weighted words;

- 2 are influential in classifying real news when present
- 0 are influential in classifying fake news when present
- 0 are influential in classifying real news when absent
- 8 are influential in classifying fake news when absent

Of the 10 lowest negative weighted words;

- 0 are influential in classifying real news when present
- 1 are influential in classifying fake news when present
- 3 are influential in classifying real news when absent
- 0 are influential in classifying fake news when absent

Therefore, there is some union between the lists in Part 3 and their counterparts in Part 6. There are no contradictions where a word predicted one classification in Part 3 and another in Part 6.

Highest Positive Weights				**Highest Negative Weights**		
#	**Word**	**Weight**		**#**	**Word**	**Weight**
1	trumps	1.84		1	hillary	-1.87
2	us	1.31		2	just	-1.29
3	says	1.26		3	victory	-1.16
4	donald	1.11		4	that	-1.10
5	turnbull	1.05		5	watch	-1.07
6	north	0.99		6	the	-1.07
7	ban	0.96		7	breaking	-1.01
8	korea	0.92		8	black	-0.96
9	australia	0.84		9	if	-0.95
10	climate	0.84		10	you	-0.95

Table 6: Highest Weights from Logistic Regression (including stopwords)

Part 6b: Highest Weights from Logistic Regression (excluding stopwords)

In comparison to 3b there is a lot of similarity in the word lists even more than the compression done in 6a. The most interesting observation is that the highest weighted words are usually most influential when they are absent rather than present.
Of the 10 highest positive weighted words;

- 2 are influential in classifying real news when present
- 0 are influential in classifying fake news when present

- 0 are influential in classifying real news when absent
- 8 are influential in classifying fake news when absent

Of the 10 lowest negative weighted words;

- 0 are influential in classifying real news when present
- 1 are influential in classifying fake news when present
- 8 are influential in classifying real news when absent
- 0 are influential in classifying fake news when absent

Highest Positive Weights				**Highest Negative Weights**		
#	**Word**	**Weight**		#	**Word**	**Weight**
1	trumps	1.84		1	hillary	-1.87
2	says	1.26		2	just	-1.29
3	donald	1.11		3	victory	-1.16
4	turnbull	1.05		4	watch	-1.07
5	north	0.99		5	breaking	-1.01
6	ban	0.96		6	black	-0.96
7	korea	0.92		7	new	-0.90
8	australia	0.84		8	rally	-0.88
9	climate	0.84		9	voter	-0.87
10	travel	0.78		10	win	-0.86

Table 7: Highest Weights from Logistic Regression (excluding stopwords)

Part 6c: Use of the Magnitudes of the Weights

The issue with using magnitudes without any sort of normalization as there can be a tendency to begin over fitting as some of the more dominate words' weights can grow out of control. This may look good on the training set but these high magnitudes will make generalization harder and can reduce performance on the test set.
For example; if the iterations from part 4 was increased to 10000 instead of 1000.

- Weight of trumps would increase from 1.84 to 3.27.
- Weight of breaking would increase from -1.01 to -2.97.
- Training set accuracy increases from 91.1% to 99.6%
- Test set accuracy decreases from 86.3% to 85.2%

As shown above, magnitudes alone can cause dominance of certain parameters which can help lower the cost function but risk over fitting. To solve this problem in Part 4. However, the magnitudes are reasonable to use in this problem as early stopping was used in Part 4.

Part 7

Part 7a: Decision Tree Classifier

Figure 2 shows an experiment where max depth values ranging from 50 to 120 are tested on the validation set to find the ideal depth or the decision tree. 70 was found to be the ideal depth when holding the other parameters constant.

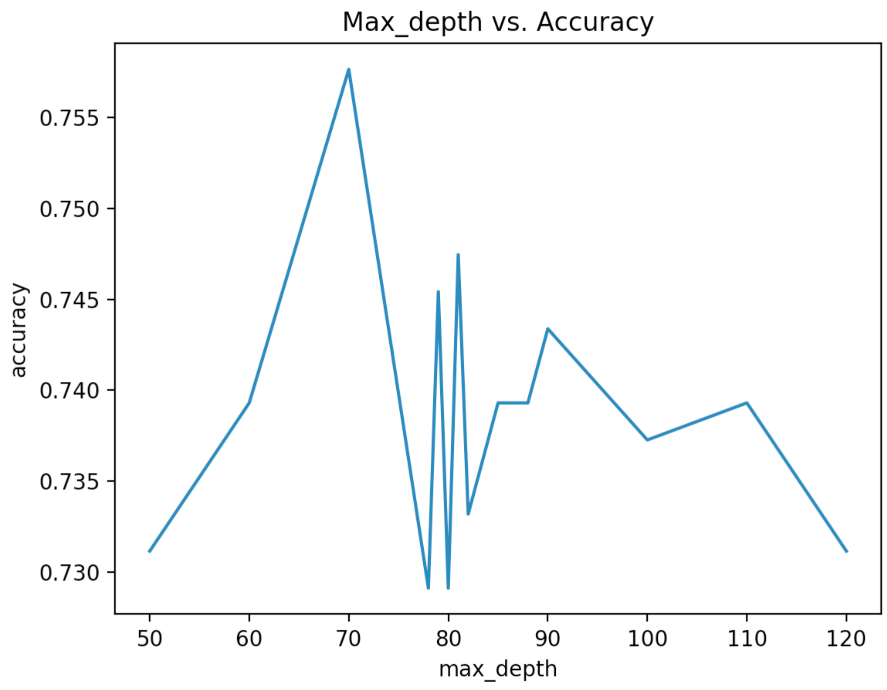

Figure 2: Validation set accuracy with respect to the max depth

More experiments were conducted to find the ideal value for both the max depth and max feature size. The final best accuracy was found when the max depth was set to 85 and the max features was set to 90. The following code shows the construction of the model.

```
clf = tree.DecisionTreeClassifier(criterion='entropy', splitter='best', \
     max_depth=85, min_samples_split=2, min_samples_leaf=1, \
     min_weight_fraction_leaf=0.0, max_features=90, random_state=None, \
     max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, \
     class_weight=None, presort=False)
clf = clf.fit(x_train, y_train)
```

Part 7b: Decision Tree Visualization

After several tests showing that the tree visualization changes every time the program is re-executed and examining that words the conclusion is that the top words are not the most influential but instead words that divide the data set most efficiently. Therefore, they have no relationship to the words shown before.

Part 7c: Method Comparison

The following table compares the three methods for classifying real and fake news. Of the methods:

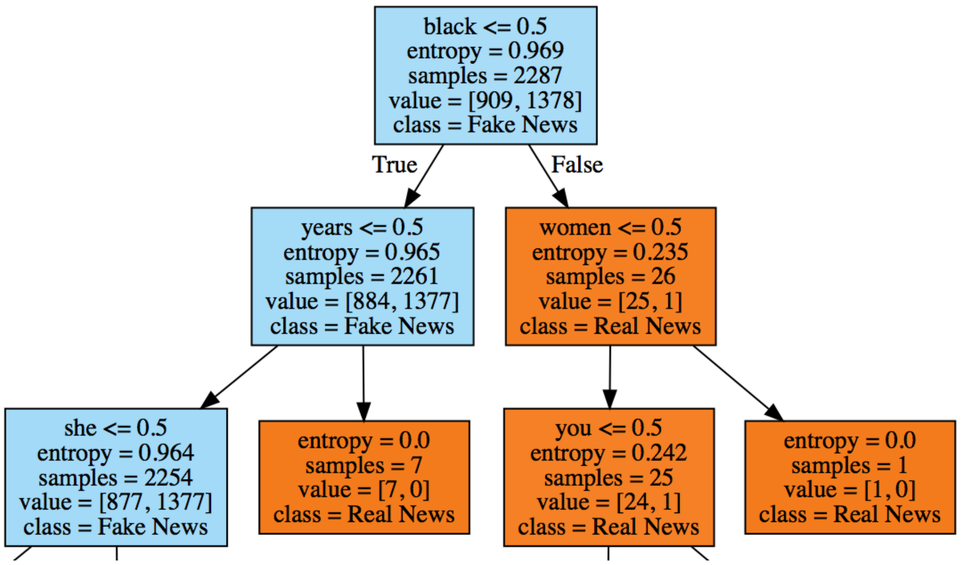

Figure 3: Visualization of the first two layers of the decision tree

Method	**Training**	**Validation**	**Testing**
Naive Bayes	93.2%	80.0%	81.5%
Logistic Regression	91.1%	82.4%	86.3%
Decision Tree	95.0%	75.9%	74.4%

Table 8: News Classifier Comparison

- Best Performance: Logistic Regression
- Worst Performance: Decision Tree
- Most Over fitting: Decision Tree

Part 8

Part 8a: Mutual Information of First Split

In this part, the mutual information of the split on the training data was computed using the decision tree from 3. $I(Y, x_i)$ is the mutual information, where Y is the random variable signifying whether a headline is real or fake, and x_i is the keyword chosen for the top most split. The calculation was done by hand and is shown below:

$$I(Y;x_i) = H(Y) - \sum_x P(X=x)H(Y|X=x) \tag{3}$$

$$I(Y;x_i=1) = H(Y) - P(X=1)H(Y|X=1) - P(X=0)H(Y|X=0) \tag{4}$$

$$= 0.969 - \frac{2261}{2287}(0.965) - \frac{26}{2287}(0.235) \tag{5}$$

$$= 0.0123 \tag{6}$$

Part 8b: Mutual Information of Second Split (for word "years")

In this part, the mutual information is calculated for a node at a lower depth in the tree.

$$I(Y;x_i) = H(Y) - \sum_x P(X=x)H(Y|X=x) \tag{7}$$

$$I(Y;x_i=1) = H(Y) - P(X=1)H(Y|X=1) - P(X=0)H(Y|X=0) \tag{8}$$

$$= 0.965 - \frac{2254}{2261}(0.964) - \frac{7}{2261}(0) \tag{9}$$

$$= 0.00398 \tag{10}$$

The mutual information gets smaller as you calculate $I(Y;x_i)$ for words lower down in the tree. This makes sense because we are gaining less information from each word in the headline. Take a headline of 10 words for example. If after 9 words, there is a probability of, say, 85% that the headline is real news. Then given a 10th word that implies the headline is fake, the probability will only be adjusted slightly as our information gain is small.