CSCI 390: Machine Learning and Cybersecurity

Fake News, Linguistics and Machine Learning

Misae Evans

Introduction

Fake News is a buzz word in the political atmosphere, mocked on social media and a growing concern for those trying to receive authentic news. There are many websites, blogs, and profiles that can maintain the appearance of a trustworthy author/organization but contain information that is either explicitly false or tweaked in pursuit of a specific goal in the realm of social engineering or seeking capital from site views. Some of the most famous fake news headlines that one might recall from 2016 are “Pope Francis shocks the world, endorses Donald Trump for president”, “Donald Trump sent his own plane to transport 200 stranded Marines”, “Ireland is now officially accepting Trump refugees from America” or “FBI director received millions from Clinton Foundation, his brother’s law firm does Clinton’s taxes”.

Social media has provided a way for anyone to voice their opinion swiftly, thoughtlessly and anonymously to an audience of millions of people giving authors of fake

news articles a massive audience. I decided to attack the problem of how to determine whether an article is fake or not. I find this to be an important social issue because of the amount of shares, views and likes these articles receive from those in the public that find them to be truthful. Fictional works should not have the power to sway public opinion in social or political affairs. If an article is labelled as “news”, the audience should be able trust the source to be factual and accurate.

There has been significant work done in the academic community in efforts to study, identify and challenge the spread of fake news. I read five different papers for background research, the following are the problems each source addressed. (1)The paper “*Automatic Deception Detection: Methods for Finding Fake News”* is where I gained most of the inspiration for my project. Linguistic assessment methods, specifically predictive deception cues were employed to identify fake news articles via the bag of words approach.(2) Classification via logistic regression and boolean crowdsourcing in effort to create automatic hoax detection systems. (3) Addressing the limited use of statistical approaches to combat fake news by increasing the the use of labeled benchmark data sets.(4)Using educational games in high schools as a vehicle to inoculate the public from fake news.(5)

Data

I found two datasets on kaggle. One data set was specifically a 2016 fake news dataset and the other was a kind of archive for legitimate news articles. In the previous deliverable I only used the fake news data set, from which I pulled 500 columns from 1,700+ columns. In the final project I used both datasets to fit and train the model to the best of my ability. I copied and pasted the data into a separate CSV file, labelled with the columns “articleType”, “text” and “ID”. The final resulting data set consisted of 1000 columns total; 500 fake articles and 500 legitimate articles.

1	type	text	uuid
2	fake	Print	6a175f46bcd24d39b3e962ad0f29936721db70db
3	fake	Why Did	2bdc29d12605ef9cf3f09f9875040a7113be5d5b
4	fake	Red State	c70e149fdd53de5e61c29281100b9de0ed268bc3
5	fake	Email Kayl	7cf7c15731ac2a116dd7f629bd57ea468ed70284

Feature Extraction

I grabbed the data from the 'text" column from the csv file above, cleaned it, transformed the cleaned data into a corpus and finally transformed the corpus into a document term matrix with the parts of speech tagged. Due to the focus of my project being linguistic analysis of fake news, I grabbed the parts of speech that I thought would vary between a fake news article and legitimate article.

The features I selected were the number of verbs, adjectives, nouns and proper nouns. After fitting my data to each model(logistic regression and random forest) I viewed the feature importances. Although many of the features turned out to not be very telling in regards to identifying a fake news article, the proper nouns had the highest percentage of importance.

Interesting Findings

I also performed topic analysis and sentiment analysis on the 2016 fake news articles, purely because I was curious. However, the findings seemed interesting enough to include in my paper. I found that the most common topics among the articles were in regards to the election, Islam or Hillary Clinton. The top five words among the articles were (1) fearing (2) fearmongering (3) fearful (4) trump and (5) fears. The polarity of the text was not too negative or positive, however the text was consistently fairly opinionated.

Analysis

The two machine algorithms used in my project were logistic regression and random forest. Both algorithms are classification algorithms. The premise of logistic regression is the idea that your input space can be sectioned off into concise and neat regions bya linear boundary, in 2D this means a straight line and in 3D this means a plane. In a random forest, each decision tree in the forest of trees considers a random subset of features and then the random forest takes an average of the individual estimates from the trees.

Cross Validation

During the cross validation process I found that the logistic regression model had an accuracy of 47% with the cross validation values ranging between 0.35 and 0.45. The random forest model had an accuracy of 52% with cross validation values ranging between 0.37 and 0.57.

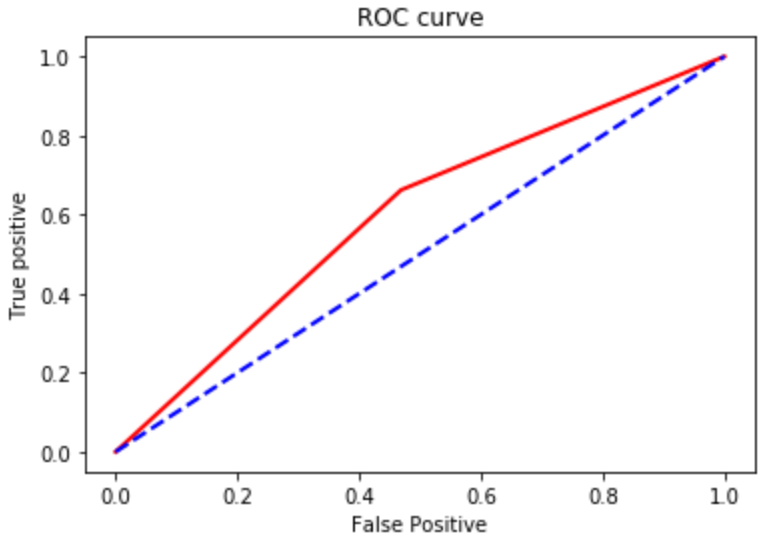
Performance Analysis

Logistic Regression

The confusion matrix for the logistic regression model displayed 154 correct assumptions.and 120 incorrect assumptions.

```
array([[ 77,  68],
       [ 52, 102]],
```

The ROC curve for the logistic regression model is displayed on the left.

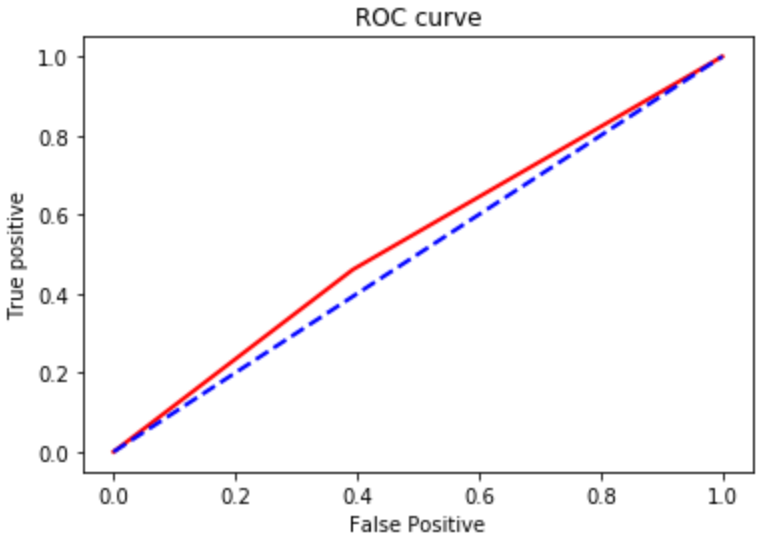


Random Forest

The confusion matrix for the random forest displayed 159 correct assumptions and 150 incorrect assumptions.

```
array([[88, 57],
       [83, 71]],
```

The ROC curve for the random forest model is displayed on the left.



Logistic Regression vs. Random Forest

Although I had my doubts throughout the course of the project, the Random Forest model turned out to be superior. The superiority in the random forest model is evident visually with the ROC curve and with the cross validation results being higher than the logistic regression model.

Conclusion

The results from the machine learning algorithms lead me to conclude that by looking at the number of verbs, adjectives, nouns and pronouns a fake news article can be identified about 52% of the time with a random forest model and about 47% of the time with a logistic regression model. Meaning, in order to predict fake news with more accuracy different features would need to be selected and more data would be necessary.

Reflection

While working independently I learned a valuable lesson in self discipline. In most of my courses I have worked in groups, I thrive in atmospheres where I can collaborate and exchange ideas with my peers. However, working independently I had to motivate myself, make a work schedule and think of creative ways to fix problems or achieve goals. I have never done work in machine learning or cybersecurity and by working independently I had the opportunity to be exposed to many areas of each subject.

Each “checkpoint” in the course of the project was helpful, it helped guide me through the course of the project and break the assignment as a whole into less intimidating pieces. In the early stages of the project I struggled with deciding which algorithm to use and identifying the differences between various algorithms. The challenge that persisted through each checkpoint was dealing with the data; finding the data, cleaning the data and breaking it down into a format I could work with was a feat to say the least.

If I had another six months I would study more machine learning algorithms and experiment with other classification models to try and find a better fit for my data. I also

would like to gather data from a different year, all of my data was limited to the year 2016. It would be interesting to see how different data sets compare to the one I collected and if the language features differ. I would also like to gather data on public opinion and see if there was a connection between fake news articles as a whole and the opinion of the american public. There are so many avenues I could explore with this project. I will definitely continue to develop this project, I am very interested to see what else I can discover.

If I could redo the project I would find a data set sooner and begin attempting to extract features, because it has been the ultimate struggle throughout the course of the semester.

Manual

First and Foremost, you are going to want to download the anaconda distribution of jupyter notebook to follow this manual exactly. I am a beginner to data science, so I did a lot of extensive research on the best "beginners" environment, jupyter seemed to be the best option for me. I haven't tried doing the same work in another environment but I am sure that the results would be the same with the latest version of python- many of the libraries used in this manual require at least python 2.7.

Data Analysis

(1) Read in the data with the pandas library- make sure its in csv format.

```
import pandas as pd
data = {}
data = pd.read_csv("fake500.csv")
data.head()
```

(2) Locate the text data from your csv file and whatever identifier you are going to use from the csv file that would make sense to you, in my case it was the UUID

```
dataToAnalyze = data['text'].values
```

```
uuid = data.iloc[:,0].values
uuid
```

(3) Put data into pandas dataframe

```
#format must be {key:[value]} for pandas
dataFixed = {key: [value] for (key, value) in pickledData.items()}
dataFixed
```

```
#  put it into a pandas dataframe
import pandas as pd
pd.set_option('max_colwidth',150)

data_df = pd.DataFrame.from_dict(dataFixed).transpose()
data_df.columns = ['text']
data_df = data_df.sort_index()
data_df
```

(4) Clean the data, below are two methods I used to clean the data, however there is no “right” way to do this and there is no specified time to stop cleaning…it is up to your needs and how far you want to go to clean it.

```
#got this method from a youtube video
import re
import string

def clean_text_round1(text):
    '''Make text lowercase, remove text in square brackets, remove punctuation and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\w*€\w*', '', text)
    text = re.sub('\w*‡\w*', '', text)
    text = re.sub('\w*¢\w*', '',text)
    text = re.sub('\w*™\w*', '', text)
    text = re.sub('\w*ð\w*', '', text)
    text = re.sub('\w*ÿ\w*', '', text)
    text = re.sub('\w*ä\w*', '', text)
    text = re.sub('\w*ä\w*', '', text)
    text = re.sub('\w*ë\w*', '', text)
    text = re.sub('\w*¯\w*', '', text)
    text = re.sub('\w*·\w*', '', text)
    text = re.sub('\w*_\w*', '', text)
    text = re.sub('\w*»\w*', '', text)
    text = re.sub('\w*á\w*', '', text)
    text = re.sub('\w*±\w*', '', text)
    text = re.sub('\w*´\w*', '', text)
    text = re.sub('\w*¶\w*', '', text)
    text = re.sub('\w*®\w*', '', text)
    text = re.sub('\w*¬\w*', '', text)

    text = re.sub('\w*«\w*', '', text)
    text = re.sub('\w*°\w*', '', text)
    text = re.sub('\w*½\w*', '', text)

    text = re.sub('\w*¦\w*', '', text)
    text = re.sub('\w*§\w*', '', text)
    text = re.sub('\w*•\w*', '', text)
    text = re.sub('\w*£\w*', '', text)
    text = re.sub('\w*‹\w*', '', text)
    text = re.sub('\w*¨\w*', '', text)
    text = re.sub('\w*„\w*', '', text)
    text = re.sub('\w*,\w*', '', text)
    text = re.sub('\w*¡\w*', '', text)
    return text

round1 = lambda x: clean_text_round1(x)
```

```
#  updated text and corpus
data_clean = pd.DataFrame(data_df.text.apply(round1))
data_clean.text
```

```
# Apply a second round of cleaning
# got this method from a youtube video
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

ps = PorterStemmer()

def clean_text_round2(text):
    '''Get rid of some additional punctuation and non-sensical text that was missed the first time around.'''
    text = re.sub(',,', '', text)

    text = re.sub(',,,', '', text)
    text = re.sub('[‘’“”…]', '', text)
    text = re.sub('\n', '', text)
    text = re.sub('aab', '', text)
    text = re.sub('aah', '', text)
    text = re.sub('aand', '', text)
    text = re.sub('aba', '', text)
    text = re.sub('abd', '', text)
    text = re.sub('abccbsfoxyounameit', '', text)
    text = re.sub('httpwwwopborgnewsseriesburnsoregonstandoffbundymilitianewsupdatesammonbundyverdictoregonstandoffmalheurcourt',
    text = re.sub('httpplanetinfowarscompoliticswhatsthetruthaboutamericanvotefraudanythingtosuspect','', text)
    text = ps.stem(text)

    return text

round2 = lambda x: clean_text_round2(x)
```

```
# updated text and corpus
data_clean = pd.DataFrame(data_clean.text.apply(round2))
data_clean.text.values
```

(5) Pickle the cleaned data so it can be used another time if desired

```
# pickle it for later use
data_df.to_pickle("corpus.pkl")
```

(6) Make a Document term matrix to get rid of stop words, tokenize the text and put it into a representation that we can work with to analyze the text. Also, we are going to pickle that document term matrix for later use.

```
# We are going to create a document-term matrix using CountVectorizer, and exclude common English stop words
#got this method from a youtube video
import nltk
#nltk.download('averaged_perceptron_tagger')
from sklearn.feature_extraction.text import CountVectorizer


cv = CountVectorizer(stop_words='english')
data_cv = cv.fit_transform(data_clean.text)
```

```
data_dtm = pd.DataFrame(data_cv.toarray(), columns=cv.get_feature_names())
data_dtm.index = data_clean.index
#data_dtm
```

```
#pickle it for later use
data_dtm.to_pickle("dtm.pkl")
```

(7) I made another document term matrix that counted the parts of speech in each text with a specialized tokenizer to help with feature selection. Pickle that one for later use as well.

```
import nltk

def tokenize(text):
    text = nltk.pos_tag(text)
    return text
    #tagged = nltk.pos_tag(text)
    #counts = Counter(tag for word, tag in tagged)
    #return counts
```

```
cv_pos = CountVectorizer(stop_words='english',tokenizer=tokenize)
data_cv_pos = cv_pos.fit_transform(data_clean.text.values)

pd.set_option('display.max_columns', 22)
data_dtm_pos = pd.DataFrame(data_cv_pos.toarray(), columns=cv_pos.get_feature_names())
data_dtm_pos.index = data_clean.index
data_dtm_pos
```

(8) Find the most common words in the text

```
from collections import Counter

words = []
for uuid in data.columns:
    top = [word for (word, count) in top_dict[uuid]]
    for t in top:
        words.append(t)

words
```

```
Counter(words).most_common()
```

```
[('fearing', 160),
 ('fearmongering', 155),
 ('fearow', 154),
 ('fearful', 153),
 ('fear', 152),
 ('fears', 147),
 ('fdot', 144),
 ('fearsit', 141),
 ('fdasolicited', 130),
 ('feasibility', 125),
 ('fcking', 121),
 ('feasible', 114),
 ('trump', 114),
 ('fbi', 109),
 ('fbis', 103),
 ('fbinavy', 92),
 ('feasiblein', 88),
 ('hillary', 86),
```

(9) Find polarity and subjectivity of text. I got this method from a youtube video and edited it to fit the needs of my project. Pickle it for later use.
    (a) Polarity: -1 is a very negative word +1 is a very positive word
    (b) Subjectivity: 0 is fact +1 is opinion

```
from textblob import TextBlob

pol = lambda x: TextBlob(x).sentiment.polarity
sub = lambda x: TextBlob(x).sentiment.subjectivity

nonSentimentData['polarity'] = nonSentimentData['text'].apply(pol)
nonSentimentData['subjectivity'] = nonSentimentData['text'].apply(sub)
nonSentimentData
```

	text	polarity	subjectivity
007c2b623b39d89ab0964f86fe0a8bd56855c0ca	There are lots of diiferent truths , when i he...	0.076587	0.357937
00a1b3f0820914870a8b3a52c6080892c05c5d9b	Source: 70News \n\nOctober 31, 2016 Updated on...	0.087653	0.463160
018d2edef81f8ac93622de17798556762119c736	DONALD TRUMP SKIPS MEDIA: Delivers Transition ...	0.142975	0.357645
01d6b7c634e6c69dd0d39c0ceacfa376b4a4e627	14 Shares\n12 0 0 2 (Taksim Square - Gezi Park...	0.009574	0.414844
0206b54719c7e241ffe0ad4315b808290dbe6c0f	Email HEALTHCARE REFORM TO MAKE AMERICA GREAT ...	0.176472	0.448784
0293466db9711d1b8e4022fa8b41e989384455d4	Found this nugget in Podesta files Fastwalkers...	0.115310	0.430825
02ae76c658c0a52244fddeaf9c14b5d0dde4e9d7	By Claire Bernish On Thursday, police from no ...	0.077778	0.372222
032d5612f4777d59ddd538980305106f90306b19	Trump should take the lead aand prempt the att...	-0.001786	0.368056
03905a0f06cb711190aad3934af253a0e1c00a83	So ,you have Rothschild banksters and British ...	-0.041667	0.208333
0548a4a7a5d414eab993531b26f053fd92e6b76f	By Nicholas West While debate surrounds the th...	0.400000	0.700000
057c1dbca3935587019a9ad2a9725b5c6bb12a97	Google "Donald Trump, pedophile"	0.000000	0.000000
063726c998dfdbe9e917b618f205b7aa7bbffcef	By Vin Armani The federal landgrab protesters ...	0.125000	0.625000
07517581586f16e842364402f43ed60f3a472858	By Brandon Turbeville As the United States mar...	0.500000	0.500000
088ddf580942133f16728aa4773e94bcc26bcfc0	Kids being made to vote for President in Schoo...	-0.018784	0.393215
095b7cc19392c3b309519cc1919657208fc25e6f	November 6, 2016 By 21wire Leave a Comment \nE...	0.242753	0.461476
096858c81b7e65acb92a0070dc7c37ac70c49a44	By Catherine J. Frompovich \nThis is the conti...	0.064802	0.439133
09a6989efa0ae4b702bf7e35b00a9902ec959485	By Bernie Suarez A recent video released by a ...	0.000000	0.250000
0ad982405c9508a9bc2383a1c04aaeeab7a1a3cf	Must be impeach of a hand page: 1 link Minglin...	0.065278	0.389583
0d526c456ad18365ca4929c2045447b59cfa109a	link originally posted by: theantediluvian It ...	0.031481	0.449074
0d599ada25a336c0df13ee689cd612154a5e9a51	By Nathaniel Mauka Congress overwhelmingly vot...	0.200000	0.508333

(10) Topic Modelling of the text to tell us what the top five topics of each article are. It should be noted that it was up to the programmer to interpret the results. I got help from a different youtube video for this method. The results tend to vary a little each time the cells are run.

```
# We're going to put the term-document matrix into a new gensim format, from df --> sparse matrix --> gensim corpus
sparse_counts = scipy.sparse.csr_matrix(tdm)
corpus = matutils.Sparse2Corpus(sparse_counts)

# Gensim also requires dictionary of the all terms and their respective location in the term-document matrix
cv = pickle.load(open("cv.pkl", "rb"))
id2word = dict((v, k) for k, v in cv.vocabulary_.items())
```

```
lda = models.LdaModel(corpus=corpus, id2word=id2word, num_topics=5, passes=10)
lda.print_topics()
```

```
2018-12-09 13:29:39,558 : INFO : topic #3 (0.200): 0.004*"serco" + 0.004*"people" + 0.003*"time" + 0.003*"trump" + 0.003*"i
nformation" + 0.003*"said" + 0.003*"new" + 0.003*"like" + 0.003*"government" + 0.003*"years"
2018-12-09 13:29:39,562 : INFO : topic #4 (0.200): 0.015*"clinton" + 0.009*"trump" + 0.007*"hillary" + 0.006*"election" +
0.005*"fbi" + 0.004*"campaign" + 0.004*"said" + 0.004*"media" + 0.004*"president" + 0.003*"new"

[(0,
  '0.005*"trump" + 0.004*"war" + 0.004*"clinton" + 0.003*"president" + 0.003*"american" + 0.003*"said" + 0.003*"new" + 0.003
*"people" + 0.003*"white" + 0.003*"states"'),
 (1,
  '0.005*"trump" + 0.004*"people" + 0.004*"isis" + 0.004*"said" + 0.004*"time" + 0.003*"white" + 0.003*"iraqi" + 0.003*"lik
e" + 0.003*"mosul" + 0.002*"just"'),
 (2,
  '0.003*"like" + 0.003*"just" + 0.003*"health" + 0.003*"time" + 0.003*"people" + 0.003*"ami" + 0.003*"smart" + 0.003*"trum
p" + 0.002*"link" + 0.002*"dont"'),
 (3,
  '0.004*"serco" + 0.004*"people" + 0.003*"time" + 0.003*"trump" + 0.003*"information" + 0.003*"said" + 0.003*"new" + 0.003
*"like" + 0.003*"government" + 0.003*"years"'),
 (4,
  '0.015*"clinton" + 0.009*"trump" + 0.007*"hillary" + 0.006*"election" + 0.005*"fbi" + 0.004*"campaign" + 0.004*"said" + 0.
004*"media" + 0.004*"president" + 0.003*"new"')]
```

(11) Call the pickled files, so we can access them to create a new dataframe based on our findings. I chose verbs, adjectives, polarity and subjectivity. We created this dataframe feed the data to the model.

```
sentiment = pd.read_pickle('sentiment.pkl')
#sentiment
```

```
polarity = sentiment.polarity.values
#polarity
subjectivity = sentiment.subjectivity.values
#subjectivity
```

```
#data_dtm_pos.to_pickle("dtmPOS.pkl")
verbs = data_dtm_pos.iloc[:,11].values
#verbs
adjective = data_dtm_pos.iloc[:,4].values
#adjective
```

```
d = {'type':articleType, 'verbs': verbs, 'adjectives': adjective, 'polarity':polarity, 'subjectivity':subjectivity }
dataframe = pd.DataFrame(data =d)
dataframe.head()
```

```
dataframe.to_pickle("dataFixed.pkl")
```

this did not work when I doubled my data size, I was forced to use the data frame below..

```
#d = {'type':articleType, 'verbs': verbs, 'adjectives': adjective, 'polarity':polarity, 'subjectivity':subjectivity }
d = {'type': articleType, 'verbs': verbs, 'adjectives': adjective, 'noun':nouns, 'properNoun':properNouns}
dataframe = pd.DataFrame(data =d)
dataframe.head()
```

	type	verbs	adjectives	noun	properNoun
0	fake	12	5	17	98
1	fake	40	43	88	575
2	fake	18	13	35	206
3	fake	100	60	128	837
4	fake	23	10	54	232

Machine Learning Algorithms

Logistic Regression Model

(1) Import the data

import data

```
import pandas as pd

#train = pd.read_csv("training.csv")
#train

train = pd.read_pickle('dataFixed.pkl')
train.head()
```

	type	verbs	adjectives	noun	properNoun
0	fake	12	5	17	98
1	fake	40	43	88	575
2	fake	18	13	35	206
3	fake	100	60	128	837
4	fake	23	10	54	232

(2) Set the X and y training values

```
#predictor_Vars= ["pronouns", "generalizingWords"]
#X,y = train[predictor_Vars], train.iloc[:,0].values
predictor_Vars=["verbs", "adjectives", "noun", "properNoun"]
X,y=train[predictor_Vars], train.iloc[:,0].values
```

(3) Right now our y training values are “fake” and “legit”, we need to encode this categorical data

```
#encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_y = LabelEncoder()
y= labelencoder_y.fit_transform(y)
```

Y before

```
y
```

```
array(['fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'fake',
```

Y after

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

(4) Split data into the training set and test set

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.cross_validation import train_test_split
#test size is gonna be the amount of data you take from overall set
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.30)
X_train
```

```
C:\Users\misae\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
n version 0.18 in favor of the model_selection module into which all the re
ote that the interface of the new CV iterators are different from that of t
  "This module will be removed in 0.20.", DeprecationWarning)
```

	verbs	adjectives	noun	properNoun
379	19	10	49	245
586	8	16	47	182
257	60	63	164	731

(5) Scale the features

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_test
#everything is on the same scale
```

```
array([[ 0.52653267,  0.0790758 ,  1.0357347 ,  0.70715098],
       [-0.63535791, -0.72912541, -0.74231262, -0.81019313],
       [ 0.13923581,  0.21842084,  0.1280602 ,  0.6282926 ],
       ...,
       [-0.41404542, -0.22748328, -0.28225842, -0.24458815],
       [-0.27572511, -0.25535228, -0.25739062, -0.06239809],
       [-0.30338918, -0.28322129, -0.10818385, -0.25002666]])
```

(6) Fit the logistic regression classifier model

```
#chosen classifier = logitistic regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

(7) View Y's predicted value

```
#fitted logistic regression on my xtrain and ytrain
#set the predicted variable (ys predicted value after the model is complete)
y_pred = classifier.predict(X_test)
y_pred
```

```
array(['fake', 'legit', 'legit', 'fake', 'fake', 'legit', 'fake', 'fake',
       'legit', 'legit', 'fake', 'legit', 'fake', 'legit', 'fake',
       'legit', 'legit', 'legit', 'legit', 'fake', 'legit', 'legit',
       'legit', 'fake', 'legit', 'fake', 'legit', 'fake', 'legit',
       'legit', 'fake', 'fake', 'legit', 'fake', 'legit', 'legit', 'fake',
       'fake', 'legit', 'legit', 'fake', 'legit', 'legit', 'legit',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'legit', 'legit',
       'legit', 'fake', 'fake', 'fake', 'legit', 'fake', 'fake', 'legit',
       'fake', 'legit', 'legit', 'legit', 'fake', 'legit', 'fake',
       'legit', 'legit', 'fake', 'fake', 'legit', 'fake', 'legit',
       'legit', 'legit', 'legit', 'fake', 'legit', 'legit', 'legit',
       'fake', 'legit', 'legit', 'legit', 'fake', 'fake', 'legit',
       'legit', 'fake', 'legit', 'legit', 'legit', 'fake', 'fake',
       'legit', 'legit', 'fake', 'legit', 'legit', 'fake', 'legit',
       'fake', 'legit', 'legit', 'legit', 'legit', 'fake', 'fake', 'fake',
       'legit', 'fake', 'legit', 'fake', 'legit', 'fake', 'fake', 'legit',
       'fake', 'fake', 'legit', 'fake', 'fake', 'legit', 'legit', 'legit',
       'legit', 'legit', 'legit', 'fake', 'legit', 'fake', 'legit',
       'legit', 'legit', 'legit', 'legit', 'legit', 'legit', 'legit',
       'legit', 'legit', 'legit', 'legit', 'fake', 'fake', 'fake',
       'legit', 'legit', 'fake', 'legit', 'fake', 'legit', 'legit',
       'fake', 'fake', 'fake', 'legit', 'fake', 'legit', 'fake', 'legit',
       'legit', 'fake', 'fake', 'legit', 'legit', 'legit', 'legit',
       'legit', 'legit', 'fake', 'legit', 'legit', 'fake', 'fake',
       'legit', 'fake', 'legit', 'fake', 'fake', 'fake', 'legit', 'legit',
       'fake', 'fake', 'legit', 'legit', 'fake', 'legit', 'fake', 'legit',
       'fake', 'fake', 'legit', 'legit', 'legit', 'legit', 'fake', 'fake',
       'legit', 'legit', 'legit', 'fake', 'legit', 'fake', 'legit',
       'fake', 'fake', 'legit', 'fake', 'fake', 'legit', 'legit', 'legit',
       'legit', 'fake', 'fake', 'fake', 'legit', 'fake', 'legit', 'legit',
       'fake', 'fake', 'fake', 'legit', 'legit', 'fake', 'legit', 'fake',
```

(8) Check out the feature importance

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
featureClf = ExtraTreesClassifier(n_estimators=50)
featureClf = clf.fit(X_train, y_train)
clf.feature_importances_
```

```
array([0.21707125, 0.19393186, 0.254797  , 0.33419989])
```

Random Forest Model

(1) Use the same X_train and y_train to fit the model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

clf = RandomForestClassifier(n_estimators=50)
clf = clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
y_pred
```

```
array(['fake', 'legit', 'fake', 'fake', 'fake', 'fake', 'fake', 'legit',
       'fake', 'legit', 'legit', 'legit', 'fake', 'legit', 'fake', 'fake',
       'legit', 'legit', 'legit', 'fake', 'legit', 'legit', 'legit',
       'fake', 'legit', 'legit', 'legit', 'fake', 'legit', 'legit',
       'fake', 'fake', 'legit', 'fake', 'legit', 'fake', 'fake', 'fake',
       'legit', 'fake', 'fake', 'fake', 'legit', 'fake', 'fake', 'legit',
       'legit', 'fake', 'legit', 'fake', 'legit', 'fake', 'legit', 'fake',
       'fake', 'fake', 'legit', 'fake', 'legit', 'fake', 'fake', 'legit',
       'fake', 'fake', 'fake', 'fake', 'fake', 'fake', 'legit', 'legit',
       'fake', 'fake', 'fake', 'legit', 'legit', 'legit', 'fake', 'fake',
       'fake', 'fake', 'legit', 'legit', 'fake', 'legit', 'legit', 'fake',
       'fake', 'legit', 'legit', 'legit', 'fake', 'fake', 'legit', 'fake',
       'fake', 'legit', 'legit', 'fake', 'fake', 'fake', 'fake', 'fake',
       'legit', 'legit', 'legit', 'fake', 'legit', 'fake', 'fake',
       'legit', 'fake', 'fake', 'legit', 'legit', 'fake', 'fake', 'fake',
       'fake', 'legit', 'legit', 'fake', 'fake', 'fake', 'legit', 'fake',
       'legit', 'fake', 'legit', 'fake', 'legit', 'legit', 'fake',
       'legit', 'legit', 'fake', 'legit', 'fake', 'fake', 'legit', 'fake',
       'legit', 'fake', 'legit', 'legit', 'fake', 'legit', 'fake', 'fake',
       'legit', 'legit', 'legit', 'legit', 'fake', 'fake', 'fake', 'fake',
       'legit', 'legit', 'fake', 'fake', 'fake', 'legit', 'legit',
       'legit', 'legit', 'legit', 'legit', 'fake', 'fake', 'fake',
       'legit', 'legit', 'fake', 'fake', 'legit', 'fake', 'fake', 'fake',
       'legit', 'fake', 'fake', 'legit', 'legit', 'legit', 'fake',
       'legit', 'fake', 'legit', 'fake', 'legit', 'fake', 'fake', 'fake',
       'fake', 'fake', 'legit', 'fake', 'legit', 'legit', 'legit',
       'legit', 'legit', 'fake', 'legit', 'fake', 'fake', 'fake', 'fake',
       'legit', 'fake', 'legit', 'legit', 'fake', 'fake', 'fake', 'fake',
       'fake', 'legit', 'fake', 'fake', 'fake', 'legit', 'fake', 'legit',
       'legit', 'fake', 'legit', 'fake', 'legit', 'legit', 'fake',
```

Performance Analysis

Logistic Regression Performance Analysis

(1) Confusion matrix for logistic regression model

```
In [22]: #make confusion matrix= the matrix that is going to show me the number of correct results and the number of incorrect results
         from sklearn.metrics import confusion_matrix

         cm = confusion_matrix(y_test, y_pred)

In [23]: #print out confusion matrix
         # 154 correct assumptions 120 incorrect assumptions
         cm

Out[23]: array([[ 77,  68],
                [ 52, 102]], dtype=int64)
```

(2) Develop the ROC curve

(a) Encode the categorical data in Y-test and Y-pred

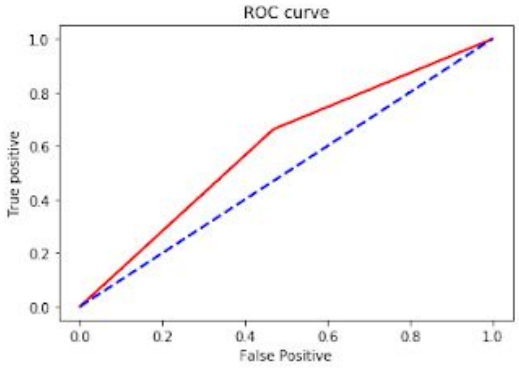
```
#encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_y = LabelEncoder()
y_test= labelencoder_y.fit_transform(y_test)
#y_test
y_pred = labelencoder_y.fit_transform(y_pred)
y_pred
```

```
array([0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0], dtype=int64)
```

(b) Visualize the ROC curve, I got tips from stack overflow and youtube to figure this out.

```
from sklearn.metrics import roc_curve
##Computing false and true positive rates
fpr, tpr,_=roc_curve(y_test, y_pred,drop_intermediate=False)

import matplotlib.pyplot as plt
plt.figure()
##Adding the ROC
plt.plot(fpr, tpr, color='red',
 lw=2, label='ROC curve')
##Random FPR and TPR
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
##Title and label
plt.xlabel('False Positive')
plt.ylabel('True positive')
plt.title('ROC curve')
plt.show()
```



(3) Cross Validation- I got this from the scikit learn website

(a) Get the cross validation scores

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(classifier, X, y, cv=5)
scores
```

```
array([0.35678392, 0.46231156, 0.49748744, 0.56281407, 0.45454545])
```

(b) Print the accuracy of these scores

```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.47 (+/- 0.13)
```

Random Forest Performance Analysis

(1) Confusion matrix, notice that instead of y_pred I used y_predict, because y predictions should be different for a different model

```
#make confusion matrix= the matrix that is going to show me the number iof correct results and the number of incorrect results
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[88, 57],
       [83, 71]], dtype=int64)
```

(2) Cross Validation Scores

```
from sklearn.model_selection import cross_val_score
scoresRF = cross_val_score(clf, X, y, cv=5)
scoresRF
```

```
array([0.37688442, 0.52261307, 0.5879397 , 0.55276382, 0.57575758])
```

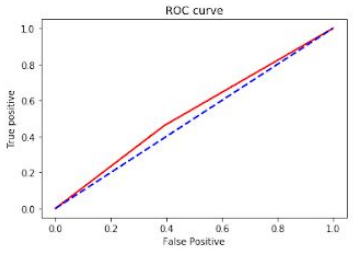
```
print("Accuracy: %0.2f (+/- %0.2f)" % (scoresRF.mean(), scoresRF.std() * 2))
```

```
Accuracy: 0.52 (+/- 0.15)
```

(3) ROC Curve

```
from sklearn.metrics import roc_curve
##Computing false and true positive rates
fpr, tpr,_=roc_curve(y_test, y_predict,drop_intermediate=False)

import matplotlib.pyplot as plt
plt.figure()
##Adding the ROC
plt.plot(fpr, tpr, color='red',
 lw=2, label='ROC curve')
##Random FPR and TPR
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
##Title and label
plt.xlabel('False Positive')
plt.ylabel('True positive')
plt.title('ROC curve')
plt.show()
```



Source Citation

Conroy, N. J., Rubin, V. L., & Chen, Y. (2015). Automatic deception detection: Methods for finding fake news. *Proceedings of the Association for Information Science and Technology*,*52*(1), 1-4. doi:10.1002/pra2.2015.145052010082

Granik, Mykhailo, and Volodymyr Mesyura. "Fake News Detection Using Naive Bayes Classifier." *2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON)*, 9 Nov. 2017, doi:10.1109/ukrcon.2017.8100379.

Roozenbeek, Jon, and Sander Van Der Linden. "The Fake News Game: Actively Inoculating against the Risk of Misinformation." *Journal of Risk Research*, 2018, pp. 1–11., doi:10.1080/13669877.2018.1443491.

Tacchini, E. (2017). Some Like it Hoax : Automated Fake News Detection in Social Networks. *Technical Report UCSC-SOE-17-05 School of Engineering, University of California, Santa Cruz*,1-12. Retrieved September 17, 2018.

Wang, W. Y. (2017). "Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection. *Department of Computer Science University of California, Santa Barbara*. Retrieved October 17, 2018.