Alternus Vera Project Paper

Team Virgo Cluster, CMPE 257
December 4, 2018

1. Introduction

Fake news today has been a big topic and concern for people in the increasingly connected society (Shu et al, 2017). Fake news has existed in the past, but has become more prevalent in the age of social networks (Shu et al, 2017).). Fake news for the purpose of discussion is not a hypothetical rumor or satire news article, but rather news articles that are designed to mislead the consumers on some topics or events (Shu et al, 2017).).

The goal of the project is to come up with a suite of Machine Learning models to generate several factors that are relevant in classifying a news articles as either fake or authentic (non-fake).

1. Introduction

2. Definitions
    2.1 News Coverage
    2.2 Political bias
    2.3 Word2Vec Embedding
    2.4 Doc2Vec Embedding
    2.5 CoVe Embedding

3. Contributions

4. Factors
    4.1 Event Coverage
    4.2 Political Bias
    4.3 Sentiment Analysis
    4.4 Context's Reliability
    4.5 Direct TF-IDF Values

5 Methods
    5.1 Data Enrichment
    5.2 Data Preprocessing
    5.3 Methods of Finding the reliability of the context of the news with Basic Scoring, LDA and Logistic
        5.3.1 Understanding the scoring the label of the news
        5.3.2 Applying LDA to extract the topics from the context

5.3.3 Classifying the consolidated vectors of context, party and topics with logistic regression
5.4 N-grams
5.5 Sentiment Analysis
5.6 Topic Modeling with Gensim
5.7 Cosine Similarity of title and text
5.8 Methods of Word Embedding by Word2Vec, Doc2Vec and CoVe
5.8.1 Word2Vec Embedding with pre-trained GloVe Word2Vec
5.8.2 Word2Vec Embedding with self-trained Gensim Word2Vec
5.8.3 Doc2Vec Embedding Using the DBOW Algorithm
5.8.4 CoVe Embedding
5.9 TF-IDF based classification
5.10 News Coverage Analysis
5.10.1 Data Enrichment
5.10.2 LDA - Analysis and Label Pairing
5.10.3 Date Normalization for Classifier
5.10.4 Word2vec for Embedding of Topics
5.10.5 Training and Testing with Random Forest
5.10.6 Final Results and Ranking
5.11. Political Bias Analysis
5.11.1 Data Enrichment
5.11.2 TF-IDF
5.11.3 Computation of Political Bias
5.11.4 The Political Bias Equation
5.11.5 Bins for Political Bias Scores
5.11.6 Training Classifier
5.11.7 Applying the Trained Classifier to Fake News Dataset
5.11.8 Using doc2vec

6. Discussions
6.1 Visualizing Topics Using Word Cloud
6.2 News Coverage Classifier

7. Limitations
7.1 Issues with various document sizes
7.2 TF-IDF on Social Media Dataset

8. Conclusions

9. Works cited

2. Definitions

2.1 News Coverage

A **News Coverage** is a number of new articles from other media outlets that covered the same latent topic depicted in the article within a predefined time window. A window of time, which is a tunable parameter, can be set to 1 month (coverage length of 30 days) and 2 months (coverage of 60 days) before and after the article was published. The actual number of related news articles cannot be determined directly, but proxy values can be obtained by leveraging a large all-inclusive news dataset.

2.2 Political bias

Political bias is a rubric indicating a range from 0 to 3 where 0 is centrist affiliation and 3 is left-most or right-most affiliation. Alternative model would be: -10 indicating the left-most affiliation and 10 being the right-most affiliation. We simply rectify this model so that values are in the positive range. This treats right-wing bias the same as left-wing bias of same magnitude of polarity.

2.3 Word2Vec Embedding

Word2Vec is an open source tool developed by google that calculates the “similarity” distances between words in a text corpus. It converts a word into a vector form based on “similarity” distances of the word to other words in the corpus, which transforms text content processing into vectorized operations. Word2Vec calculates the cosine similarities, and the distance range is 0-1. The larger the value, the higher the correlation between the two words. As a result, semantic similarities of the text are represented (or so called “embedded”) by distributed similarities in the vector spaces.

2.4 Doc2Vec Embedding

Although Word2Vec is able to embed semantic information between words into compressed vector dimensions, many times we need to get the vector representations of sentences and the entire document in order to capture more contextual information. **Doc2Vec** is designed to achieve this goal. Like Word2Vec, Doc2Vec has two models: Distributed Memory (DM) and Distributed Bag of Words (DBOW). The DM model predicts the probability of a word given a context and a document vector, and the DBOW model predicts the probability of a set of random words in a document given a document vector. Comparing the models between Word2Vec and Doc2Vec, Doc2Vec's DM model is similar to Word2Vec's CBOW, whereas Doc2Vec’s DBOW model is similar to Word2Vec's Skip-gram.

2.5 CoVe Embedding

Similar to Doc2Vec, **Context Vectors** (CoVe) provides vectorized embedding to represent contextual semantics of sentences and the entire document. Unlike Doc2Vec, the CoVe model is trained based on a deep 2-layer bidirectional LSTM encoder to better capture contextual information into contextualized word vectors.

3. Contributions

Hyunwook Shin has contributed to the following work: 1) enriching the dataset using all-news (non-fake) articles for training Machine Learning models and social media dataset specifically for political bias analysis 2) looked into and leveraged word2vec for news coverage analysis and doc2vec and political bias analysis 2) applied TF-IDF vectorization to train and apply political bias scoring model 3) preprocessed raw text, leveraging team members' python snippets, tailoring and refactoring code for distillation, 4) As part of distillation, leveraged LDA analysis to extract topics for news coverage analysis, along with sentiment analysis and 5) applied various classifiers such as Naive Bayes and Random Forest models to classify or predict political bias scores and news coverage scores as part of the ranking. Hyunwook is also responsible for organizing the paper, writing the outline, joining pieces from the team members by writing introductions and transitions. Hyunwook was responsible identifying important factors, assigning weightage, and hosting custom generated .csv files.

Yu Xu has contributed to the following works: 1) enriched the original dataset; 2) researched on Word2Vec , Doc2Vec and CoVe embedding methods; 3) applied pre-trained and self-trained Word2Vec models; 4) applied self-trained Doc2Vec model; 5) applied pre-trained CoVe model; 6) Tuned hyperparameters for various models to reach an optimal performance of ~ 86% accuracy.

Mojdeh Keykhanzadeh has contributed to the following works : 1) Originally researched about IBM fairness and did Distillation(remove punctuation);2) Apply ngrams(applied trigram);3) Sentiment analysis using both VADER and NaiveBayesClassifier;4) Visualizing most frequent words in the text using wordcloud; 5) Topic modeling using Gensim;6) Finding cosine similarity between text and title to find consistency of text and title 7)Also responsible for applying sentiment to each row of text/title and prepared csv for deriving polynomial . (Note that I did apply same approach for topic/text consistency and cosine similarity factors to add to my csv columns but did not have succes sun after 4 hours because of slow environment)

Lin Cheng has contributed to the following works: 1) data Interpretation and data transformation for fake news classification; 2) Apply TF-IDF to get vectorised doc-word matrix for further modeling; 3) Apply different classify algorithms and turn the models, found the best accuracy 95.9% with weighted LogisticRegression; 4) Tried LDA for topic modeling.

Sy Le has contributed to the following works: data imports, initial works for stemming, lemmatization, and tokenizers. On top of this, Sy also picked up exploring context as the factor to indicate fake news. He also applied LDA, and Count Vectorizer to apply logistic regression to classify if a piece of news is fake.

4. Factors

4.1 Event Coverage

The main motivation behind **event coverage** is the hypothesis that there is a positive correlation between outside coverage of the latents event depicted in the article and the credibility and ultimately truthfulness of the article as a whole. In other words, the more news sources (outside the article) talk about the same event, the more credible the article is as a whole. Some fake news are designed to confuse audiences and make them lose trust in reliable news (Shu et al, 2017).

We as readers do this from day to day when we read sensational or eye-catching articles. When we read an article about a topic we think is "too good to be true", we cross-check with other news websites under Google News or Twitter to read up on the event or topic. This is backed by the idea that there are fake users and bot accounts purposely designed to spread false stories (Shu et al., 2017).

Through tips and advice on slack, just because there are ample number of coverage on some topic, it does not necessarily mean that the particular article in question is a reliable (non-fake news). (Arsanjani, 2018). This factor should be used in conjunction with other factors such as "bias", "sensationalism", and "social media activities" to further reduce the impurity of the classifications. We can treat this factor as an intermediary step to be factored early in the process to remove True Positives.

Nevertheless, a Random Forest model, constructed with LDA and word2vec vectorization methods, was able to give a good accurate estimate of the coverage scores on unseen articles.

4.2 Political Bias

The motivation behind the **political bias** is the hypothesis that the articles that are neutral (or centrist) have higher probability of being truthful compared to articles that are to the far ends of the spectrums (either right-wing or left-wing).

0	1	2	3
Neutral	Slightly Partisan	Partisan	Strongly Partisan

4.3 Sentiment Analysis

Sentiment analysis was applied to text and title to find out author's attitude toward a topic and categorize it . To achieve this, we first used VADER library to find polarity scores of sentences in the document (both text and titles). With VADER , the text was splitted into smaller chunks of sentences and generated compound , neutral , negative and positive scores for each chunks.

4.4 Context's Reliability

The main reason why context is extremely important when it comes to identify the integrity of any statements is that certain medium like a TV ad which are could be fully sponsored by a particular party could potentially distort the integrity of a fact. TV Ads are not reliable because they are paid for to convey a message that potentially be beneficial to their sponsors. In this context, we should proceed with cautions.

Reliability of the context is defined by extracting the keywords out of context along with the party affiliation. Then we assign a reliability score for each of these context by matching the context against a list of word we have hand picked. This score ranges from 0 to 10.

4.5 Direct TF-IDF Values

TF-IDF values, unlike other factors such as sentiment or political bias have no intuitive meaning on its own. However, TF-IDF values are considered to be very important factor in determining whether the news article is either real or fake. As a result, TF-IDF was applied directly on the Kaggle fake news dataset for the purpose of classifying whether a given news article is real or fake.

By training and verifying TF-IDF model one can use the TF-IDF as a vectorized (non-scalar) factor in determining whether a news article is fake or not. It is important to note that TF-IDF is also used internally to come up with Machine Learning model for political bias factor as well.

Hypothetical Table Composed of TF-IDF Score and Other Factors

Row	Sentiment	Coverage	Political Bias	Reliability	TF-IDF [Col 1]	TF-IDF [Col 2]	...	TF-IDF [Col N]
0	4	8	3	5	0.001	0.010	...	0.0033
1	2	6	1	6	0.04	0.001	...	0.0022
...	...	...	...		...	..	...	...

5 Methods

5.1 Data Enrichment

The original Kaggle dataset (https://www.kaggle.com/mrisdal/fake-news) is heavily skewed to fake news (13,000 news with labels of: 88% ‘bs’, 3% ‘bias’ and 8% ‘others’). In order to train a good fake news prediction model, a reliable "non-fake" news dataset containing 50,000 source verified non-fake news was introduced (https://www.kaggle.com/snapcrack/all-the-news). In different analyses, we either added a subset (i.e. 10,000) or the entire 50,000 “non-fake” news on top of the original 13,000 news to build up the dataset.

Another dataset composed of political social media posts is also chosen for political bias analysis (Eight, 2016).

5.2 Data Preprocessing

Words in the text column, which contains texts information of each news, were tokenized and transformed into list of tokens per entry (document) via the following data preprocessing steps:

- Data grouping and re-labeling
- Text tokenization
- Converting word tokens to lower case
- Removal of stop words and punctuations
- Removal of English words
- Word stemming and lemmatization (including pros and cons)
- Removal of non-alphabetic tokens and short tokens with length < 3

5.3 Methods of Finding the reliability of the context of the news with Basic Scoring, LDA and Logistic

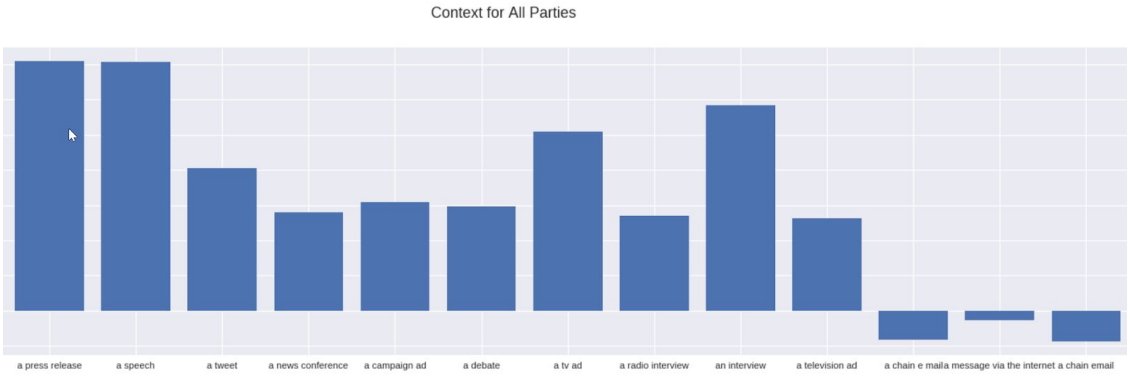
5.3.1 Understanding the scoring the label of the news

We have a total of 6 different output labels. So we applied a fixed number according to the output labels as followed:

- True: 1
- False: 0
- Half-true: 0.5
- Pants-fire: -1
- Barely-true: 0.6
- Mostly-true: 0.8

While the scores above are mostly self explanatory, one area that should be stressed is that “pants-fire” is assigned a negative score because that’s when the news is totally wrong and has a negative impact on the authenticity of the news.

Then we sketch basic bar chart to find out the context of the news that has the most output score given the above formula. We found that “a press release” or “a speech” has the most score whereas the news has lowest reliability score when it happens as “a message on the internet”



After that, we drilled down further and looked at the scoring given a particular party affiliation, and find the overall score when the party is democrat or republican. A good conclusion we find here is that republican party tweets tend to be more reliable than democrat party tweets.

5.3.2 Applying LDA to extract the topics from the context

Here we used gensim to do LDA with ngram equals 2 to extract the topics out of context and the news statements. The reason here is that we want to preserve the context and difference between a “campaign ad” vs “tv ad”.

5.3.3 Classifying the consolidated vectors of context, party and topics with logistic regression

We constructed the matrix consisting of the Countvectorizer of Topics from the Statements, Party, and the Context of the news. Then we used logistic regression with the above features and train our model to classify if a news is fake or not.

5.4 N-grams

To identify certain words occurring together for better visualization , N-grams was used . The following all techniques of N-grams were tried but trigram were applied a final delivery :

- Unigram

- Bigram
- Trigram

5.5 Sentiment Analysis

One finding was that it is important to remove punctuation while doing sentiment analysis in our case to avoid counting punctuation as emotion .The sum of positive, negative and neutral scores should equal to 1 and compound score is normalized score between -1 as most extreme negative and +1 as most extreme positive .
In the following snippet of code, polarity_scores() method from VADER library is used to achieve the above sentiment . Note that for our analysis, we only looked at first 10 titles/text to reduce computation. Lets see the snippet code for analysing titles:

```
def sentiment_analyzer_scores(sentences):

    for sentence in sentences[:10]:
      print(sentence)
      score = sid.polarity_scores(sentence)
      for key in sorted(score):
         print('{0}: {1}, '.format(key, score[key]), end='')
      print()

sentiment_analyzer_scores(sentences_kg_title)
```

And the result came up as following :

```
muslim busted stole million gov benefit
compound: 0.4588, neg: 0.0, neu: 0.625, pos: 0.375,
attorney general loretta lynch plead fifth
compound: 0.0, neg: 0.0, neu: 1.0, pos: 0.0,
breaking weiner cooperating fbi hillary email investigation
compound: 0.0, neg: 0.0, neu: 1.0, pos: 0.0,
pin drop speech father daughter kidnapped killed isi voted donald j trump 100percentfedup com
compound: -0.765, neg: 0.375, neu: 0.625, pos: 0.0,
fantastic trump 7 point plan reform healthcare begin bombshell 100percentfedup com
compound: 0.5574, neg: 0.0, neu: 0.714, pos: 0.286,
hillary go absolutely berserk protester rally video
compound: 0.0, neg: 0.0, neu: 1.0, pos: 0.0,
```

The above result obviously showed us that the authors of second and third title who are “Barracuda Brigade” and “reasoning with facts” and are 100% neutral . The same analysis were applied to the text to see the attitude of author in the text as well in order to compare the result for better analysis. The result was slightly different but still showed like in the above case still second and third author are more neutral .

The following is a part of result of applying same analysis on text which shows second and third author neutral position :

```
print pay back money plus interest entire family everyone came
compound: 0.2023, neg: 0.149, neu: 0.671, pos: 0.18,
attorney general loretta lynch plead fifth barracuda brigade 20
compound: -0.7783, neg: 0.122, neu: 0.797, pos: 0.082,
red state fox news sunday reported morning anthony weiner coope
compound: 0.9661, neg: 0.051, neu: 0.746, pos: 0.203,
```

To get overall text polarity scores and try some other techniques , we used classification with training and prediction by NaiveBayesClassifier .Basically , we classified text based on training set .
Creating a dictionary of positive , negative and neutral words was the first step followed by using bag of words , converting dictionary to features . Our train set was set as sum of all features which was used to train classifier .
Here is the summary snippet of above step :

```
#creating dictionary ,defining 3 classes : positive, negative and neutral
positive_vocab = [ 'wow','awesome', 'outstanding', 'fantastic', 'terrific', 'good',
                   'nice', 'great' ,'Thanks','winner']
negative_vocab = [ 'bad', 'terrible','useless','horrific','hate','fuck']
neutral_vocab = [ 'will','the','say','was','is','did','know','words','not','yes' ]

#using bag of words , convert word  to features
def word_feats(words):
    return dict([(word, True) for word in words])

positive_features = [(word_feats(pos), 'pos') for pos in positive_vocab]
negative_features = [(word_feats(neg), 'neg') for neg in negative_vocab]
neutral_features = [(word_feats(neu), 'neu') for neu in neutral_vocab]

#make train set , the sum of the 3 features
train_set = negative_features + positive_features + neutral_features

#train classifier
classifier = NaiveBayesClassifier.train(train_set)
```

Finally , the result came up as following :

```
Positive: 0.9525348103700285
Negative: 0.0343872605585045
Neutral: 0.013077929071467036
```

Based on results, we found more positivity on the whole text. Although VADER provided better analysis for each chunk of sentences but this was another way of finding sentiment by defining

our own dictionary accuracy of our model in case we combined all results from VADER and compare the compound scores with above scores .

5.6 Topic Modeling with Gensim

To identify which topic is discussed in the document , we used Gensim library . As a part of Gensim , LDA (latent Dirichlet Allocation) is used as topic modeling technique along with converting our dictionary which was created from data to bag-of-words corpus.
The following lines defines the above concept .
Convert to bag-of-words:

```
from gensim import corpora
dictionary = corpora.Dictionary(data_kg_fake_news ['text_clean_tokenize'])
corpus = [dictionary.doc2bow(text) for text in data_kg_fake_news ['text_clean_tokenize']]
```

Using LDA:

```
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = NUM_TOPICS, id2word=dictionary, passes=15)
```

The following is the result of applying to 5 topics .

```
(0, '0.009*"wa" + 0.006*"one" + 0.005*"people" + 0.004*"like"')
(1, '0.017*"clinton" + 0.016*"trump" + 0.012*"wa" + 0.011*"hillary"')
(2, '0.031*"de" + 0.019*"la" + 0.012*"в" + 0.010*"en"')
(3, '0.009*"u" + 0.008*"ha" + 0.006*"state" + 0.006*"government"')
(4, '0.014*"u" + 0.012*"russia" + 0.009*"russian" + 0.009*"war"')
```

The above showed us the weightage associated with each topic which was found in the text . Trump and clinton seems to be dominant here as well . To compare the result of text with title , we applied the same method on titles and the result is as follows.

```
(0, '0.044*"nan" + 0.026*"trump" + 0.007*"voting" + 0.005*"vote"')
(1, '0.014*"de" + 0.010*"la" + 0.006*"un" + 0.006*"human"')
(2, '0.008*"year" + 0.008*"u" + 0.008*"war" + 0.007*"police"')
(3, '0.040*"hillary" + 0.039*"clinton" + 0.019*"election" + 0.014*"fbi"')
(4, '0.042*"trump" + 0.015*"russia" + 0.012*"u" + 0.011*"donald"')
```

Still Trump and Clinton shows as major topics .

5.7 Cosine Similarity of title and text

Exploring another way of finding consistency between title and text , finding cosine similarity score of title and text got our attention . Cosine similarity between sentences in title and

sentences in text as two vectors was computed . To achieve this , we required to convert text to vector. Snippet of code is as follows .

```
def get_cosine(vec1, vec2):
     intersection = set(vec1.keys()) & set(vec2.keys())
     numerator = sum([vec1[x] * vec2[x] for x in intersection])

     sum1 = sum([vec1[x]**2 for x in vec1.keys()])
     sum2 = sum([vec2[x]**2 for x in vec2.keys()])
     denominator = math.sqrt(sum1) * math.sqrt(sum2)

     if not denominator:
        return 0.0
     else:
        return float(numerator) / denominator

def text_to_vector(text):
     return Counter(text)

vector1 = text_to_vector(sentences_kg)
vector2 = text_to_vector(sentences_kg_title)

cosine = get_cosine(vector1, vector2)
```

The result showed a score of around 0.26 which was quite rational as title has less wording than text . To improve this we did apply cosine similarity for topics of text/title for each row of news .

5.8 Methods of Word Embedding by Word2Vec, Doc2Vec and CoVe

5.8.1 Word2Vec Embedding with pre-trained GloVe Word2Vec

A pre-trained Word2Vec model containing uncased 400K vocabulary with 50-dimension vectors was downloaded from Global Vectors for Word Representation (GloVe: https://nlp.stanford.edu/projects/glove). Tokens in each document were first converted to Word2Vec vectors. This resulted a vectorized dataset with dimensions of *m* X *n* X *v*, where *m* represents numbers of news (documents), *n* represents numbers of words per document (note: n varies as words in a document vary), and *v* represents Word2Vec vector length for each word. This matrix was then aggregated by ‘mean’, ‘median’, ‘max’ and ‘min’ to *m* X *v* in order to remove the various factor of document lengths. After concatenation, the final matrix with a shape of *m* X *4v* was fed to Machine Learning modeling as the input.

```
def transform(self, X):
    if self.method=='mean':
        return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for words in X
        ])
```

5.8.2 Word2Vec Embedding with self-trained Gensim Word2Vec

An un-trained Word2Vec model from Gensim was used. Other steps are the same as running a pre-trained Word2Vec model.

5.8.3 Doc2Vec Embedding Using the DBOW Algorithm

An un-trained Doc2Vec DBOW model from Gensim with a vector size of 300 was used. First a Doc2Vec vocabulary was built using word tokens from the entire dataset. The Doc2Vec DBOW model was then trained through 30 epochs to obtain word vectors for each document. It was noted that Doc2Vec DBOW would automatically pad each document to have the same length as the vocabulary size, by assigning zero to vocabulary words that did not appear in the corresponding document. Therefore, unlike Word2Vec the aggregation step became unnecessary for Doc2Vec. Finally the resulting matrix with a shape of *m* X *vocabulary_size* was fed to Machine Learning modeling as the input

```
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, min_count=1, alpha=0.065, min_alpha=0.065)
model_dbow.build_vocab([x for x in tqdm(doc2vec_all_data)])
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(doc2vec_all_data)]),
                     total_examples=len(doc2vec_all_data), epochs=1)
```

5.8.4 CoVe Embedding

A pre-trained CoVe model based on a 2-layer bidirectional LSTM deep learning model was used (https://github.com/rgsachin/CoVe/blob/master/PortFromPytorchToKeras.ipynb). This CoVe model was wrapped around Doc2Vec vectors built in the previous section (see "*Doc2Vec Embedding Using the DBOW Algorithm*") in order to further capture contextual information via training. Finally the resulting matrix with a shape of *m* X *vocabulary_size*2* was fed to Machine Learning modeling as the input.

```
from keras.models import load_model
# init cove model
# Cove model is saved in 'Keras_Cove.h5'
cove_model = load_model('./glove.6B/Keras_CoVe.h5')
# CoVe asks for 3-dim (batch_size, doc_size, vector_size)
X_train_cove_ = np.expand_dims(train_vectors_dbow,axis=0)
# Encode doc2vec vector by CoVe
X_train_cove_ = cove_model.predict(X_train_cove_)
```

Modeling by Sci-kit-learn Machine Learning Models and Hyperparameter Tuning

The resulting input matrices from Word2Vec, Doc2Vec and CoVe vectorization were first splitted by training and test sets. The training set was trained on various Sci-kit-learn models (*i.e.* logistic regression, svm, random forest, gradient boost and extreme gradient boost) and

accuracies were compared based on predictions on the test set. The best model was further tuned by grid-searching hyperparameters of the model.

Model Performance

The best performing model is a logistic regression model with the following hyperparameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'saga'}. It is based on Doc2Vec DBOW with an accuracy of 86%.

5.9 TF-IDF based classification

From the analysis of the data, there are ~13000 Kaggle fake news dataset with labels. 443 articles with bias label, 11492 articles with bs label, 430 with conspiracy labels, 19 labelled as fake, 246 labelled as fake, 102 labelled as junksci, 146 labelled as satire, 121 labelled as state. There is also a separate set of 50,000 articles from enriched verified non-fake news data. Because the number of the fake news articles is low, we marked all of the fake news as “bs”, and non-fake news sample as “non-bs”. By reducing the number of labels, we avoid unnecessary noises which could result in bias result.

```
type
bs       12999
non-bs   50000
```

After initial relabeling. TD-TDF is applied using TfidfVectorizer with SVD. Here a common mistake to avoid is that do not use “fit_transform” on test documents, as the model is trained on training documents.

```
vectorizer = TfidfVectorizer(tokenizer=tokenize2)
svd_model = TruncatedSVD(n_components=200,
                         algorithm='randomized',
                         n_iter=10)
svd_transformer = Pipeline([('tfidf', vectorizer),
                            ('svd', svd_model)])
svd_transformer=vectorizer

vectorised_train_documents = svd_transformer.fit_transform(X_train)
vectorised_test_documents = svd_transformer.transform(X_test)
```

With the vectorised term-doc matrix, we can apply different classifications on it. We tried using RandomForest, SVM and LogisticRegression with cross validation and grid search. Eventually we found that weighted LogisticRegression gave the best result, since we have much more “non-bs” samples, so weighting “bs” a little heavier worked the best.

```
logistic = LogisticRegression(class_weight={"bs":5,"non-bs":3})
C = [0.1, 1]
penalty = ['l1','l2']

param_grid = dict(C=C, penalty=penalty)
gs = GridSearchCV(logistic, param_grid=param_grid, cv= 5, scoring='accuracy')

gs.fit(vectorised_train_documents, y_train)
print(gs.best_params_)

y_pred=gs.predict(vectorised_test_documents)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("F1:",metrics.f1_score(y_test, y_pred, pos_label='bs'))
print(metrics.confusion_matrix(y_test, y_pred))
```

```
{'C': 1, 'penalty': 'l1'}
Accuracy: 0.9566137566137566
F1: 0.8913619501854796
[[ 3364   419]
 [  401 14716]]
```

We can now use our model to check news, the one below is from Bloomberg:

```
test_data = np.array(["Had President Donald Trump been successful in launching p
It never happened, apparently thwarted by then-White House Counsel Don McGa
RELATED: Trump raised prosecuting Clinton with top White House, Justice officials
At the very least, the latest developments underline how Trump's senior subordina
And it leaves anyone on the outside wondering what other potential disasters top
They also raise questions about the capacity of a now-understaffed White House a
It will be impossible to confirm, given the habitual silence from the special counsel'
That will play into rising tensions in Washington amid expectations that the endgar
Bombshell reports by CNN and The New York Times about the President's intentio
s_test = pd.Series(test_data)

vectorised_test_documents = svd_transformer.transform(s_test)
y_pred=gs.predict(vectorised_test_documents)
print(y_pred)
```

```
['non-bs']
```

5.10 News Coverage Analysis

5.10.1 Data Enrichment

In order to assign coverage score to each article of the fake news dataset, a rich database of news articles spanning multiple years of data is required. One option is to leverage public APIs such as NewsAPI or Google Trends API. While using public API may give more accurate results, doing so can subject training step to artificially low rate-limits (number of requests is

limited using free accounts). Instead, a rich dataset from Kaggle known as “All News Dataset” is imported into the notebook, and use as the “database”.

Kaggle's "All the news" dataset was used to curate reference articles to enrich the study of coverage analysis (Thompson, 2017). The relevant dataset files are downloaded from Kaggle, unzipped and re-uploaded to GitHub repository (Thompson, 2017).

```
dock2 ~/wses/all_news_dataset_kaggle$ unzip articles1.csv.zip
Archive:  articles1.csv.zip
  inflating: articles1.csv
dock2 ~/wses/all_news_dataset_kaggle$ unzip articles2.csv.zip
Archive:  articles2.csv.zip
  inflating: articles2.csv
dock2 ~/wses/all_news_dataset_kaggle$ unzip articles3.csv.zip
Archive:  articles3.csv.zip
  inflating: articles3.csv
```

The csv files are almost 200MB in size. Therefore, we must treat them differently before uploading to GitHub.

```
dock2 ~/wses/all_news_dataset_kaggle$ git lfs track *.csv
"articles1.csv" already supported
"articles2.csv" already supported
"articles3.csv" already supported
...
dock2 ~/wses/all_news_dataset_kaggle$ git remote add origin
git@github.com:h7shin/all_news_dataset_kaggle.git
...
dock2 ~/wses/all_news_dataset_kaggle$ git push -u origin master
Uploading LFS objects:   0% (0/1), 54 MB | 4.5 MB/s
```

Since "articles1.csv" was sufficient to get a good coverage analysis. The data from articles2.csv and articles3.csv were dropped. Including all data unnecessarily, can result in slow distillation, training and testing.

After data is downloaded, basic steps are taken to preprocess the information, such as tokenization, lemmatization, removal of stop words, before moving the important parts of distillations, as shown below.

5.10.2 LDA - Analysis and Label Pairing

First, LDA analysis is performed to label each rows of the fake news dataset and “All News” dataset. The coefficients in front of each word are dropped as part of simplification. The assumption is that the top two words comprising the topic are both significant enough to be treated equally. It is important that the goal is to build a reliable prediction model. While there is

a risk of oversimplification, if the final model results in a poor accuracy score, the coefficient can always be reintroduced.

Second, to assign coverage score, the topic “label” for each fake news article is searched in the “all news” dataset. For each label matches, we increment the coverage score by 1. However, if the publication dates of the two news articles (one from fake news article, and one from the “all news” dataset) do not fall in the same time window (30 days or 60 days), no points are awarded to the coverage score from that particular match. If no news articles match, the coverage score would be assigned a score of 0 accordingly.

```
import datetime

print( "Start Time", datetime.datetime.now() )

def coverage( article ):
    fromdate, todate = window( article[ 'date' ], 15 )
    selected_coverage = all_kaggle[(all_kaggle['date'] > fromdate) & (all_kaggle['date'] < todate)]
    selected_coverage['covered'] = selected_coverage.apply( lambda r: r[ 'topics' ][0] in article.topics and
                                                            r[ 'topics' ][1] in article.topics, axis=1 )
    return len(selected_coverage[selected_coverage['covered'] == True])

data_kaggle[ 'coverage' ] = data_kaggle.apply( coverage, axis=1 )

print( "Finished Time", datetime.datetime.now() )
```

Illustration of Label Pairing to Compute Coverage (Mock Data for Illustrative Purpose Only)

Table A (10 articles)

Articles	Topic Words	Computed Coverage Score	Rationale
1	"Apple", "Banana"	0	Matches None from B
2	"Apple", "Orange"	1	Matches B.2
3	"Orange", "Pair"	2	Matches B.3 and B.1023
...	...		

Table B (10,000 articles)

Articles	Topic Words
1	"Apple", "Pair"
2	"Apple", "Orange"
3	"Orange", "Pair"
...	...
1023	"Orange", "Pair"
...	...

Sentiment analysis has also been performed on the kaggle dataset.

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
0    {'neg': 0.135, 'neu': 0.669, 'pos': 0.196, 'co...
1    {'neg': 0.129, 'neu': 0.815, 'pos': 0.057, 'co...
2    {'neg': 0.051, 'neu': 0.791, 'pos': 0.158, 'co...
3    {'neg': 0.3, 'neu': 0.46, 'pos': 0.24, 'compou...
4    {'neg': 0.093, 'neu': 0.722, 'pos': 0.185, 'co...
5    {'neg': 0.356, 'neu': 0.591, 'pos': 0.053, 'co...
6    {'neg': 0.135, 'neu': 0.723, 'pos': 0.142, 'co...
7    {'neg': 0.193, 'neu': 0.735, 'pos': 0.072, 'co...
8    {'neg': 0.066, 'neu': 0.842, 'pos': 0.093, 'co...
9    {'neg': 0.152, 'neu': 0.792, 'pos': 0.056, 'co...
Name: sentiment, dtype: object
```

5.10.3 Date Normalization for Classifier

A new column called “date_int” was created by converting publication date of format **“YYYY-MM-DD**” to “**YYYYMM**” format of type integer. Day of the month is dropped to minimize overfitting and total number of classes. It should be noted that while exact date is needed to compute the *actual* coverage score, the predicted coverage score during supervised learning does not necessarily need exact date since the coverage score is already calculated. Conversion from date to integer format is important to train the classifier.

5.10.4 Word2vec for Embedding of Topics

Using **word2vec**, the vectorized embeddings of each topic terms are generated. Since we label each row with two words comprising the topic (e.g. “apple”, “price”), two vectors are generated per row. Since the dimension of each vector is 100, number of topic terms is 2, and one “date_int” column is used, the total of 100 x 2 + 1 = 201 columns are generated as input for training the classifier model. TF-IDF approach was also considered, but due to very sparse matrix generated by TF-IDF, it was dropped in favor of word2vec.

5.10.5 Training and Testing with Random Forest

Random forest with 10 trees was trained to predict the coverage score using the 201 factors computed using simplified publication date (Section 5.10.3), and remaining 200 columns from word2vec vectors of the latent topics (Section 5.10.4). After training the model, a classifier performed well on the test date with accuracy score around 90% ~ 95% (Section 6.2).

5.10.6 Final Results and Ranking

Please refer to the final discussion sections (6.2).

5.11. Political Bias Analysis

5.11.1 Data Enrichment

For political spectrum/bias we focus on whether the article is politically biased from 0 being centrist/neutral and 10 being right-most or left-most. The key is to use supervised learning to determine whether the political pieces are democratic or republicans based on the words.

We use the following dataset from Kaggle, (Eight, 2016)
https://www.kaggle.com/crowdflower/political-social-media-posts

Pre-populated columns of the social media posts to come up with the political bias score.

5.11.2 TF-IDF

The aim is to leverage the partisan or neutral rating of each social media post as foundation of our supervised learning. TF-IDF analysis is performed to convert texts from each social media posts to vectors.

5.11.3 Computation of Political Bias

```
def BiasEncoder( bias ):
   if bias == 'neutral':
      return 0
   elif bias == 'partisan':
      return 1

def MessageEncoder( message ):
   return {
         'attack' : 12,          # Rationale: attacking another politician clear indication of high political (partisanship) score
         'mobilization' : 6,     # rationale: designed to mobilize supports, clear indication of high political score
         'support' : 5,          # Rationale: message of politcal support is a good indication of high political score
         'policy' : 4,           # Rationale: discussion of political policy is a modest indication of high political score
         'media' : 3,            # Rationale: Interaction with media has some indication of high political score
         'other' : 3,            # Rationale: We don't know (either way)
         'constituency' : 2,     # Rationale: Discussion of political constintuency is a poor indication of high political score
         'personal': 1,          # Rationale: Condolences and sympathy are clear indication of low political score
         'information' : 0 }[ message ]

def AudienceEncoder( audience ):
   return {
        'constituency' : 2,      # Rationale: Address to political constituency has higher political/partisanship score than address to nati
        'national' : 1
        }[ audience ]

political_data['biasNumeric'] = political_data.bias.apply(BiasEncoder)
political_data['messageNumeric'] = political_data.message.apply(MessageEncoder)
political_data['audienceNumeric'] = political_data.audience.apply(AudienceEncoder)
```

Bias Encoding Table

Type	Weight
Neutral	0
Partisan	1

Message Encoding Table

Category	Weight	Rationale
attack	12	attacking another politician is a clear indication of partisanship
mobiliation	6	designed to mobilize supports, clear indication of partisanship
support	5	message of political support is a good indication of partisanship
policy	4	discussion of political policy is a modest indication of partisanship
media	3	Interaction with media has some indication of partisanship
other	3	We don't know (either way)
constituency	2	Discussion of political constituency is an indication of neutrality
personal	1	Condolences and sympathy are clear indication of neutrality
information	0	Presentation of information is likely neutral

Audience Encoding Table

Type	**Weight**	**Rationale**
Constituency	2	Addressing to specific political audience is likely partisan
National	1	(see above)

5.11.4 The Political Bias Equation

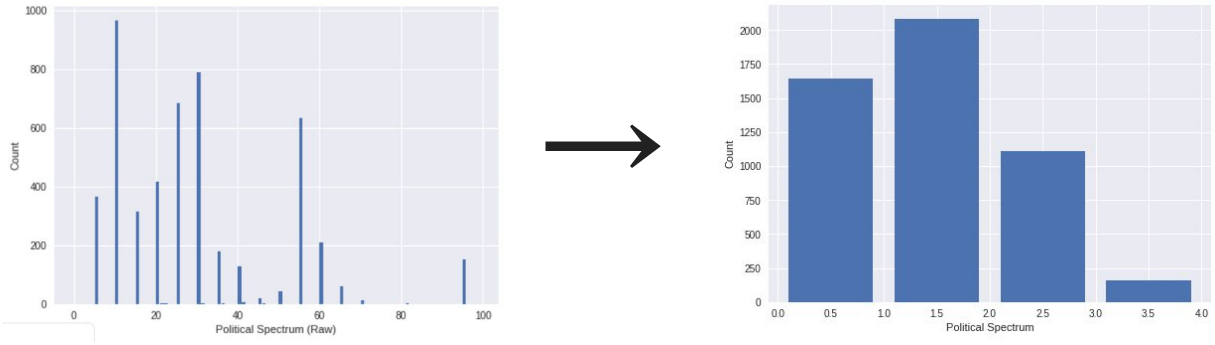
$$\textbf{\textit{Political Bias}} = \textbf{\textit{Bias}} \times 30 \times \textit{Confidence} + \textbf{\textit{Message\_Type}} \times 5 + \textbf{\textit{Audience}} \times 5$$

We gave more weight to “bias” value, since as the name suggest, it is the most direct factor that indicates political bias. Confidence score, which is also provided in the dataset, is used to attenuate the weight of the bias column. The weight of the “Message Type” can vary from 0 points to 60 points. The message labeled as “attack” is a very strong indicator of partisanship, whereas messages labelled as “information” or “personal” indicates absence of partisanship. The audience score is given relatively low weight given relatively weaker connection between the target audience and political bias.

The weights are adjusted after multiple iteration of training and testing on the enriched data. We also ensure that political bias is scaled from minimum value of 0 to maximum value of 100.

5.11.5 Bins for Political Bias Scores

After examining the distribution of scores, the boundaries are chosen to allocate article to each bin evenly (and minimize boundary scores). With just raw scores from 1 to 100, accuracy score can be poor (TF-IDF + MultinomialNB). As a result, a continuous range is reduced to a set to four discrete classes (0 - 3) . Verifying the distribution, boundary ranges **[0-20)**, **[20-40)**, **[40-80)**, and **[80-100)** are chosen.

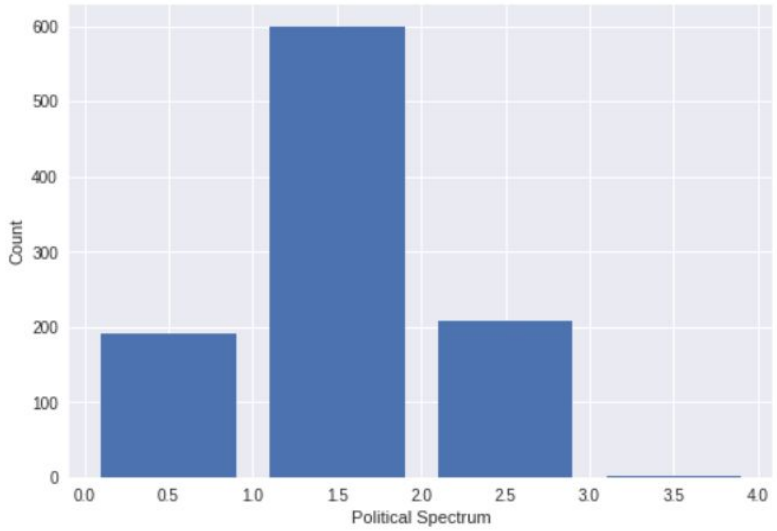


5.11.6 Training Classifier

Prior to using TF-IDF classifier, I ran basic preprocessing steps (lemmatization, tokenization, removal of stopwords. Two classifiers are considered, namely count-vectorizer and TF-IDF. Given TF-IDF values as input and the political bias scores as output, the Multinomial Naive Bayes classifier is trained on 66% of the dataset. The accuracy score of the classifier on the remaining dataset (test data) hovered around 44 ~ 66 percent.

5.11.7 Applying the Trained Classifier to Fake News Dataset

The classifier is then used to assign political bias scores to each row of the fake news dataset from Kaggle. The distribution of the political bias score is shown below:



5.11.8 Using doc2vec

Using term frequency approach (TF-IDF) we got many entries with very sparse vectors. Because the resultant vectors are difficult to distinguish we have very low resultant accuracy score. word2vec may give us more dense vectors to work with. However, word2vec is specific to individual words. To apply at document/text level we should use doc2vec. Let us use doc2vec to see if we get better prediction model. Accuracy score with doc2vec with random forest is around 40%, as shown below

5.12. Data Aggregations

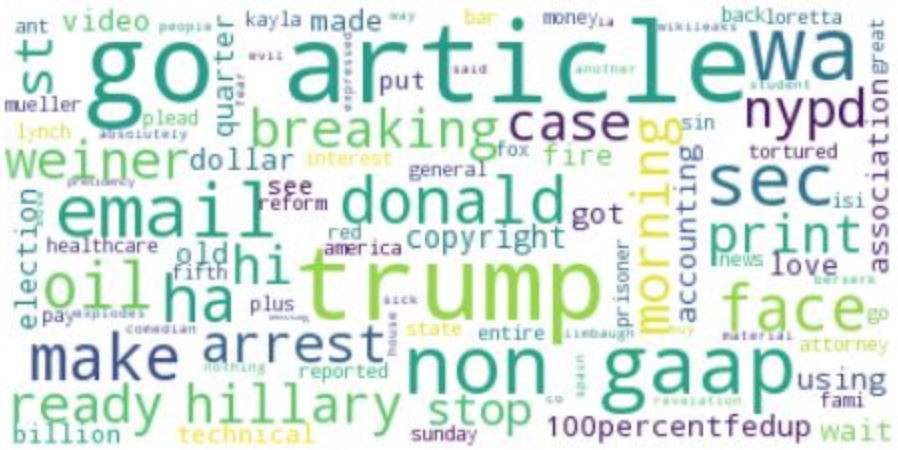
Some of the factors researched have been aggregated and used to predict whether the news is fake or not. To do this, each members of the team is responsible for distilling and ranking each fake news entry (based on some factor) in collective dataset composed of both fake and non-fake news, prepared by Gene. After labeling the data with computed factor scores, we were responsible for upload the data in csv format, then combine the csv data into one large “master” dataset.

After populating the master dataset with scores each corresponding to a factor, we trained random forest classifier and compared the importance of factors. From those scores, we generated weights for each factors. Standardization of scales also needed to be taken into account as well.

6. Discussions

6.1 Visualizing Topics Using Word Cloud

Frequency of words in our text was visualized before moving on to topic modeling , we found out the frequency of words using wordcloud library. Note that maximum number of words to output is set to 100 for easier visualization. As shown in the figure, below the topic keywords such as “Donald Trump” and “Hillary” were the quite prevalent.



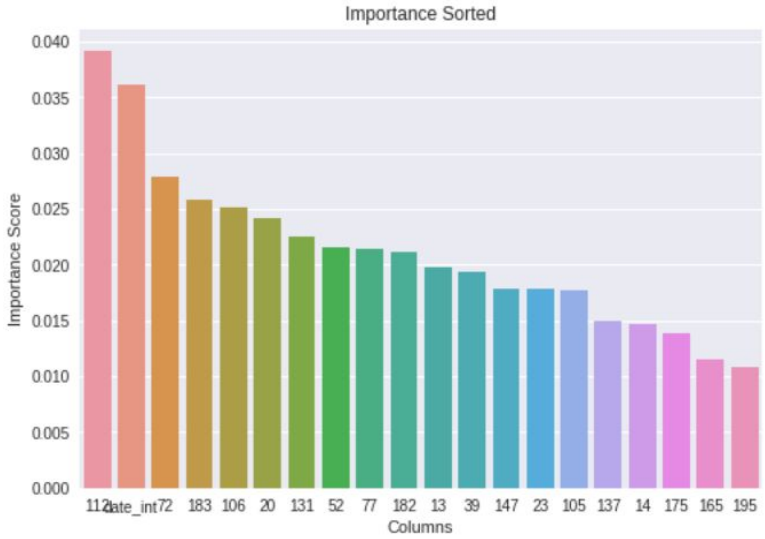
6.2 News Coverage Classifier

Using the steps taken in the Methods section, a classifier with accuracy score around 90% ~ 95% was created.

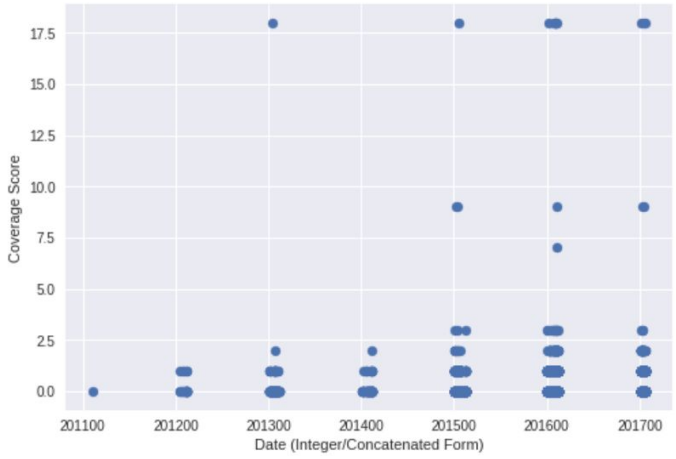
```
1 from sklearn import metrics
2
3 'Accuracy', metrics.accuracy_score(y_test, y_pred)
```

```
('Accuracy', 0.9333333333333333)
```

As shown in the following figure, "date_int" is the one of the most important features in determining whether or not the news has high news coverage. This is expected since the date the news has been posted definitely impacts on whether the news has sufficient coverage from peer articles. Some of the numeric columns have significant impact on the prediction of the coverage score. But, since these factors are latent, they do not correspond to specific words in the vocabulary.



THe same model has been applied to generate labels for each article under combined dataset (kaggle fake news dataset and all news dataset). THe scatter plot of indicating the date published (in YYYYMM form) and the coverage values are shown below for illustration.



7. Limitations

7.1 Issues with various document sizes

A common problem encountered during applying Word2Vec is that the size of each document varies. After converting texts to word vectors through Word2Vec embedding, the vectors vary in lengths. Numpy will not convert these various length vectors to ndarray. As a result, many

numpy and pandas functions can no longer be used (*e.g.* transformer transform function, estimator fit and predict functions).

There are a few solutions to this issue: 1) Brute-force padding, which is computationally expensive; 2) Keras has a *"pad_sequence"* functional API that can pad vectors to the same length. 3) Doc2Vec by default will convert and pad the document to the same size as the vocabulary size. 4) Sci-kit-learn *sklearn.feature_extraction.text* module has several vectorization function APIs (*e.g.* CountVectorizer, TfidfVectorizer) that can also by default convert and pad the document to the same size as the vocabulary size. 5) Instead of aggregating word vectors by individual word, one can aggregate across document size for each document (see "*Word2Vec Embedding* with pre-trained GloVe Word2Vec"). The major drawback of this approach is that it would lose a lot of informations when aggregating across the document, which could be to some extent mitigated by concatenating multiple aggregation sections (*e.g.* min, max, mean, median, quantile).

- Though stemming is super fast when we clean up the sentences, we find stemming to causes a lot of truncation and transformed the english words into unreadable and not human friendly words.
- When we tried to do LDA on context of the topics using ngram=1, we find the result to be fairly skewed. During this extraction process of ngram=1, we partially lose the meaning of the sentence. For example, a "radio interview" is really different from a "TV interview"

7.2 TF-IDF on Social Media Dataset

Based on the observation, the TF-IDF vectors used in the social media post were very sparse, resulting in weak prediction model. One possible reason why TF-IDF was not a good approach was that the length of the text for each row was rather small. If the text size is small, TF-IDF cannot distinguish stop words and important words accurately. If the model is to be improved, other alternatives such as word2vec can be explored.

8. Conclusions

We have explored various factors in determining whether or not a given article is fake news or not. Using random forest classifier, we determined which factors are the most important factors in determining. Preliminary analysis has shown that social reliability factors have been the strongest indicator of fake news. Title /text topic consistency analysis along with cosine similarity gave us away to find out if text is relevant to its title.

9. Works cited

1. Arsanjani, A. (2018, November 18). Notes : [Notes under #weekly-notes on slack]. Retrieved November 20, 2018.
2. Brownlee, J. (2017, October 6). How to Develop Word Embeddings in Python with Gensim. Retrieved November 27, 2018, from https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
3. Budhiraja, A. (2018, May 14). A simple explanation of document embeddings generated using Doc2Vec. Retrieved December 4, 2018, from https://medium.com/@amarbudhiraja/understanding-document-embeddings-of-doc2vec-bfe7237a26da
4. Calculate cosine similarity given 2 sentence strings. (2017, December 12). Retrieved November 27, 2018, from https://stackoverflow.com/questions/15173225/calculate-cosine-similarity-given-2-sentence-strings
5. Classifier comparison¶. (n.d.). Retrieved November 20, 2018, from https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
6. Drozd, N. (2016, May 20). DS lore. Retrieved November 25, 2018, from http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/
7. Drozd, N. (2017, December 13). Nadbordrozd/blog_stuff (M. Beckmann marcelobeckmann & N. Duedil, Eds.). Retrieved November 25, 2018, from https://github.com/nadbordrozd/blog_stuff/blob/master/classification_w2v/benchmarking_python3.ipynb
8. Eight, F. (2016, November 20). Political Social Media Posts. Retrieved November 20, 2018, from https://www.kaggle.com/crowdflower/political-social-media-posts
9. FakeNewsChallenge. (2017, June 14). Retrieved November 27, 2018, from https://github.com/FakeNewsChallenge
10. How to overcome training example's different lengths when working with Word Embeddings (word2vec). (2016, August 1). Retrieved November 27, 2018, from https://datascience.stackexchange.com/questions/13120/how-to-overcome-training-examples-different-lengths-when-working-with-word-embedding
11. Hulstaert, L. (2017, November 13). LDA2vec: Word Embeddings in Topic Models – Towards Data Science. Retrieved November 27, 2018, from https://towardsdatascience.com/lda2vec-word-embeddings-in-topic-models-4ee3fc4b2843
12. Le, Q. V., & Mikolov, T. (n.d.). Distributed Representations of Sentences and Documents. Retrieved November 25, 2018, from https://cs.stanford.edu/~quocle/paragraph_vector.pdf.
13. Li, S. (2018, March 30). Topic Modelling in Python with NLTK and Gensim – Towards Data Science. Retrieved November 27, 2018, from

https://towardsdatascience.com/topic-modelling-in-python-with-nltk-and-gensim-4ef03213cd21

14. Li, S. (2018, May 31). Topic Modeling and Latent Dirichlet Allocation (LDA) in Python. Retrieved November 20, 2018, from https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
15. Li, S. (2018, May 31). Topic Modeling and Latent Dirichlet Allocation (LDA) in Python. Retrieved November 27, 2018, from https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
16. Li, S. (2018, September 17). Susanli2016/NLP-with-Python. Retrieved November 25, 2018, from https://github.com/susanli2016/NLP-with-Python/blob/master/Doc2Vec Consumer Complaint_3.ipynb
17. Ma, E. (2018, October 20). Replacing your Word Embeddings by Contextualized Word Vectors. Retrieved November 26, 2018, from https://towardsdatascience.com/replacing-your-word-embeddings-by-contextualized-word-vectors-9508877ad65d
18. Martin, B., & Koufos, N. (n.d.). Sentiment Analysis on Reddit News Headlines with Python's Natural Language Toolkit (NLTK). Retrieved December 8, 2018, from https://www.learndatasci.com/tutorials/sentiment-analysis-reddit-headlines-pythons-nltk/
19. McCann, B., Bradbury, J., Xiong, C., & Socher, R. (n.d.). Learned in Translation: Contextualized Word Vectors. Retrieved November 25, 2018, from https://arxiv.org/pdf/1708.00107.pdf.
20. News API. (n.d.). News API - A JSON API for live news and blog articles. Retrieved November 20, 2018, from https://newsapi.org/
21. Pandey, P. (2018, September 23). Simplifying Sentiment Analysis using VADER in Python (on Social Media Text). Retrieved November 27, 2018, from https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f
22. RaRe-Technologies. (2018, October 4). RaRe-Technologies/gensim. Retrieved December 4, 2018, from https://github.com/RaRe-Technologies/gensim/blob/develop/gensim/models/doc2vec.py
23. Rehurek, R. (2018, September 20). Gensim: Topic modelling for humans. Retrieved December 4, 2018, from https://radimrehurek.com/gensim/models/doc2vec.html
24. Risdal, M. (2016, November 25). Getting Real about Fake News. Retrieved November 20, 2018, from https://www.kaggle.com/mrisdal/fake-news
25. Sachithanandam, G. (2018, April 02). Rgsachin/CoVe. Retrieved November 26, 2018, from https://github.com/rgsachin/CoVe
26. Saldaña, Z. W. (2018, January 15). Sentiment Analysis for Exploratory Data Analysis. Retrieved November 27, 2018, from https://programminghistorian.org/en/lessons/sentiment-analysis

27. Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake News Detection on Social Media. ACM SIGKDD Explorations Newsletter, 19(1), 22-36. doi:10.1145/3137597.3137600
28. Shu, Kai, Bernard, Russell, H., Liu, & Huan. (2018, April 26). Studying Fake News via Network Analysis: Detection and Mitigation. Retrieved November 25, 2018, from https://arxiv.org/abs/1804.10233
29. Thompson, A. (2017, August 20). All the news. Retrieved November 20, 2018, from https://www.kaggle.com/snapcrack/all-the-news
30. Two-letter words | Oxford Dictionaries. (n.d.). Retrieved November 27, 2018, from https://en.oxforddictionaries.com/explore/two-letter-words
31. Vu, D. (2018, August 7). Generating WordClouds in Python. Retrieved November 27, 2018, from https://www.datacamp.com/community/tutorials/wordcloud-python