Identifying Ads in TV News Closed Captions

By Linnea Hartsuyker

Problem Statement

TV News closed captioning captions the ads as well as the news itself. This closed caption text is a rich source of information about what news stories are being covered and what language is used to discuss them, but when it's contaminated with ads, it's harder to extract information about the news itself.

On the other hand, knowing who advertises on which show is also useful information for media watchdog groups, so being able to identify ads is an important processing step in doing natural language processing on TV News closed captions

Data

The data will be downloaded from archive.org, which maintains a huge repository of TV news closed captions.

Data Gathering

I modified the download scripts from https://github.com/notnews/archive_news_cc to help me download TV News Closed Captioning from archive.org. I followed the following steps:

1. Download the list of all available tv shows
2. Use that list to build two lists, one of all FOX News shows, and one of all CNN shows
3. Feed those lists into the scraper that downloads HTML and XML files, one for each show

Data Parsing

I wrote code to parse metadata from the XML files including:

- Contributor: TV affiliate
- Runtime: the length of the program in HH:MM:SS
- Start_time: the datetime when the program started
- End_time: the datetime when the program ended
- Subject: a list of subjects covered in the program

The HTML from each show divides the closed captioning text into 60-second snippets, so I parsed:

- Snip_start: the start of the snippet in seconds since the beginning of the program
- Snip_end: the end of the snippet in seconds since the beginning of the program
- Snippet: the text of the snippet

This is combined with the metadata and output to a CSV of snippets, one for CNN and one for FOX.

Please find the code to do this downloading and parsing here:
https://github.com/LinneaHarts/ad_finder_cc/tree/master/src/data

Data Cleaning and Feature Creation

One of the biggest challenges with NLP classification is coding the text. I was that unsupervised learning could help make the task of coding sentences of closed captioning as ad or news easier.

Before experimenting with vectorizing and clustering, I used NLTK to tokenize the snippets into sentences and combined sentences of fewer than 3 words into the previous sentence. I removed all punctuation and some other special characters.

Then I tried several different approaches to using unsupervised learning to (a) help with hand-coding and (b) create features that could help with machine learning. My process was as follows:

1. Using TFIDF, lemmatize and then vectorize the sentences into words and bi-grams that could be used in a variety of bag-of-words analyses
2. Use Kmeans to cluster the data into 75 clusters. I then printed out representative sentences from these clusters and sorted them into ad, news, or mixed. While this could not replace hand-coding, since the vast majority of sentences sorted into one mixed cluster, I was able to identify ad clusters and news clusters among the other sentences.
3. I used the Kmeans cluster assignment to create a rough logistic regression model.
4. I applied this model to snippets to create a feature called snip_ad, which was 1 if the rough model identified an ad
5. I added features for kmeans clusters and whether I had identified that cluster as ad, news, or mixed
6. Using the topic modeling from before, I added the topic scores to the features, so each sentence is scored by how well it fits into any of 75 topics. While these topics don't sort ads particularly well, I hoped they would be useful features for a supervised machine learning model
7. With visual inspection of the data, I noticed certain words, like "next", "welcome back", "applause" and others might indicate ads in the following or previous rows, so I created features like **next_welcome** and **prev_next** among others to capture the presence of those words in nearby sentences.

After this I hand-coded 10,000 sentences of CNN closed captioning and 10,000 sentences of FOX News closed captioning for use in training supervised machine learning models.

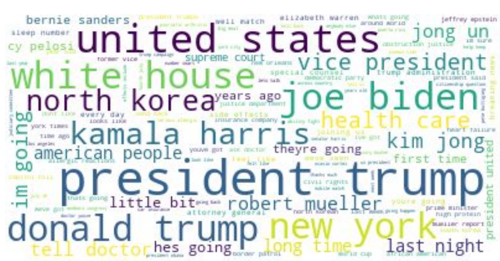
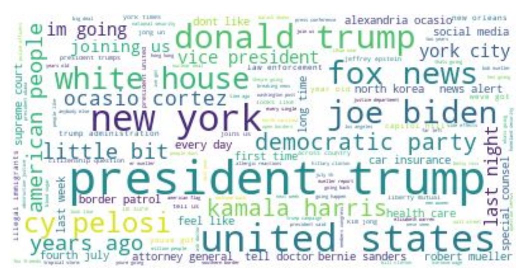
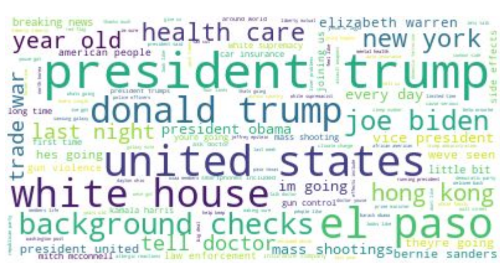
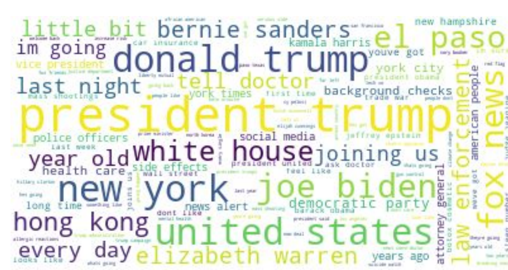
Initial Findings

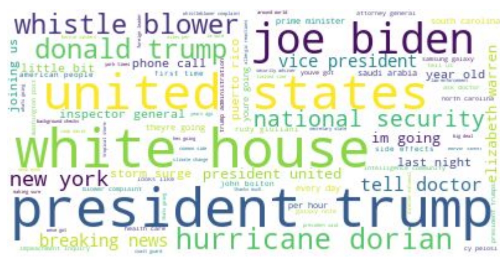
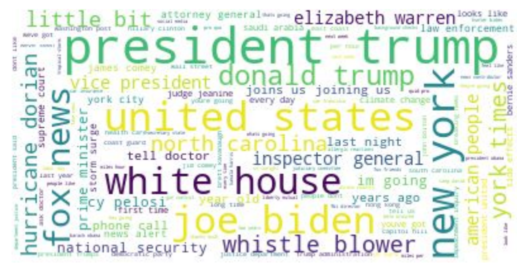
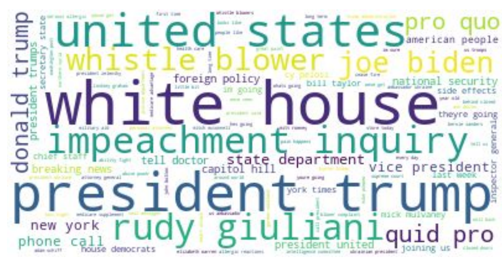
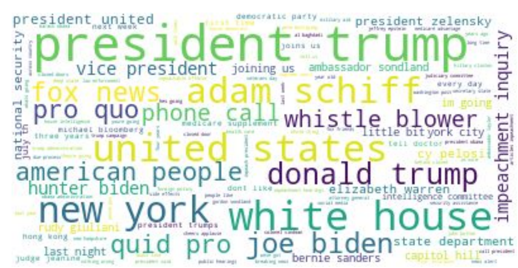
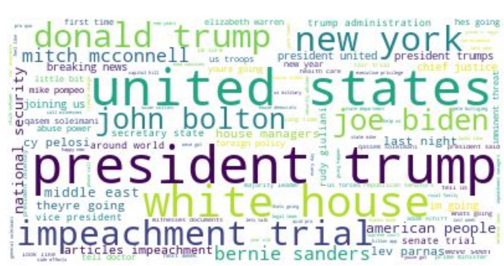
Exploratory Data Analysis

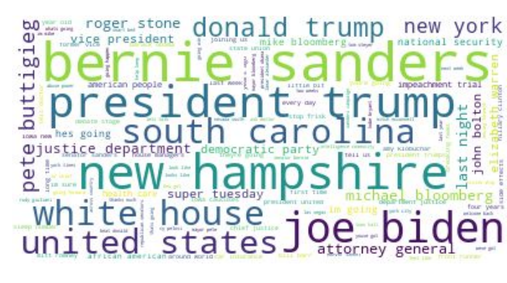
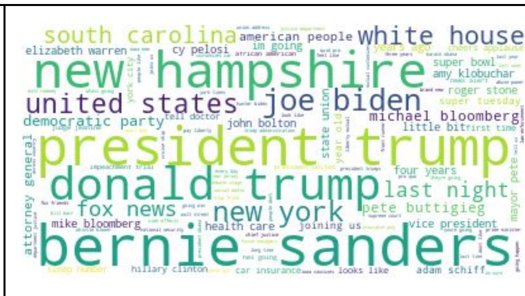
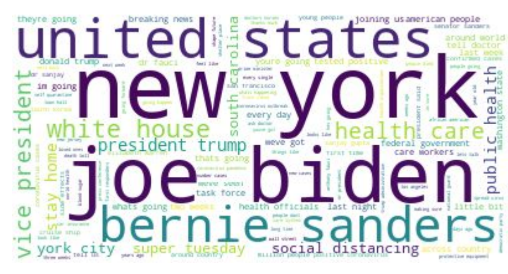
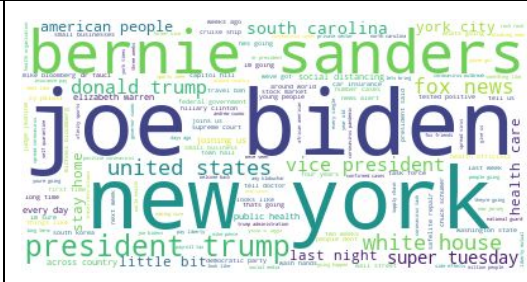
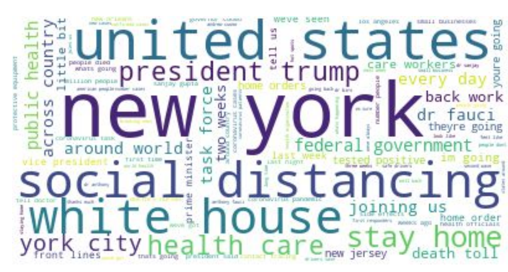
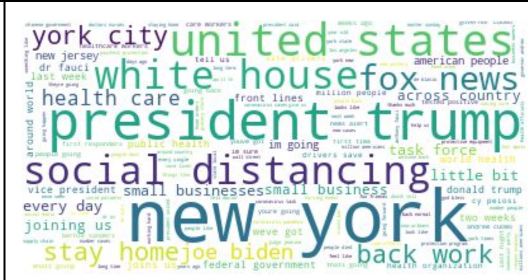
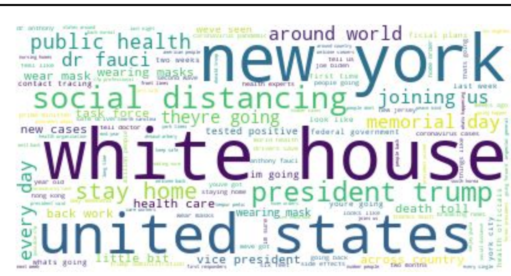
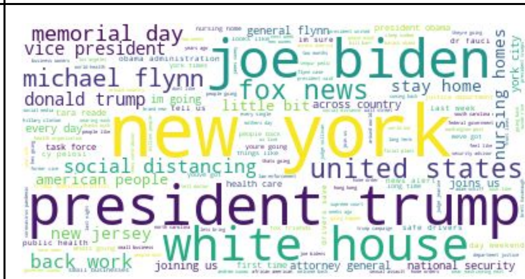
Exploratory analysis can be less than useful for NLP projects, but I still used some tools to visualize what I could.

Word Clouds

I used to explore to visualize the top bi-grams for FOX and CNN over the past year.

<table>
<tr><th>Month</th><th>CNN</th><th>FOX News</th></tr>
<tr><td>June 2019</td><td>
</td><td>
</td></tr>
<tr><td>July 2019</td><td>
</td><td>
</td></tr>
<tr><td>August 2019</td><td>
</td><td>
</td></tr>
</table>

September 2019		
October 2019		
November 2019		
December 2019		
January 2020		

<table>
<tr><td>February 2020</td><td>
</td><td>
</td></tr>
<tr><td>March 2020</td><td>
</td><td>
</td></tr>
<tr><td>April 2020</td><td>
</td><td>
</td></tr>
<tr><td>May 2020</td><td>
</td><td>
</td></tr>
</table>

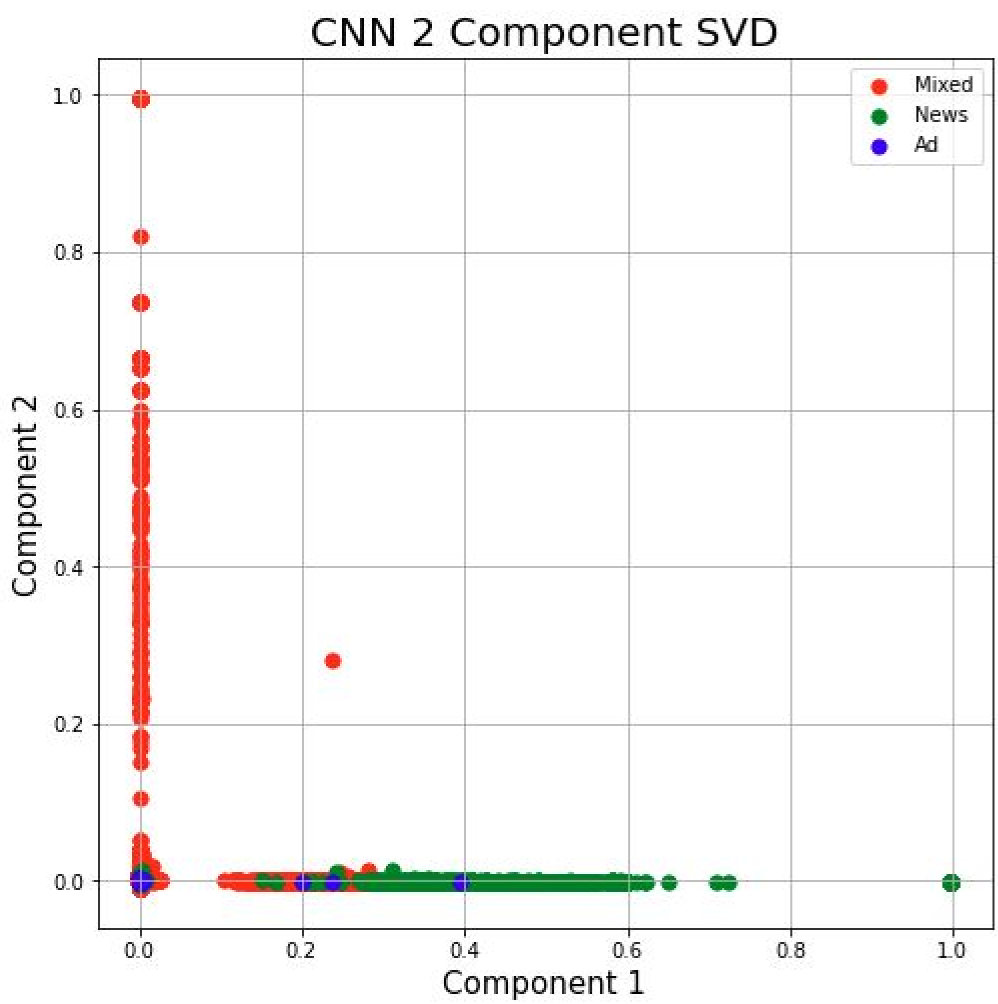
While many months did not have much difference, since keywords from news stories tended to dominate, it's interest to note a few things:

- August 2019 shows “background checks” along with “el paso” for CNN, showing that CNN was talking about solutions to gun violence, while FOX was report on it but not talking solutions
- In January 2020, CNN was talking about John Bolton, who had a tell-all book about Trump coming out, but FOX did not cover it as much
- In May 2020, CNN was still talking about Dr. Fauci, public health, and social distancing since the COVID-19 epidemic was still ongoing, while FOX was pushing stories about Michael Flynn

The code to create the CNN word clouds is here: https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/CNN%20Word%20Clouds.ipynb and the FOX word clouds here: https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/FOX%20Word%20Clouds.ipynb

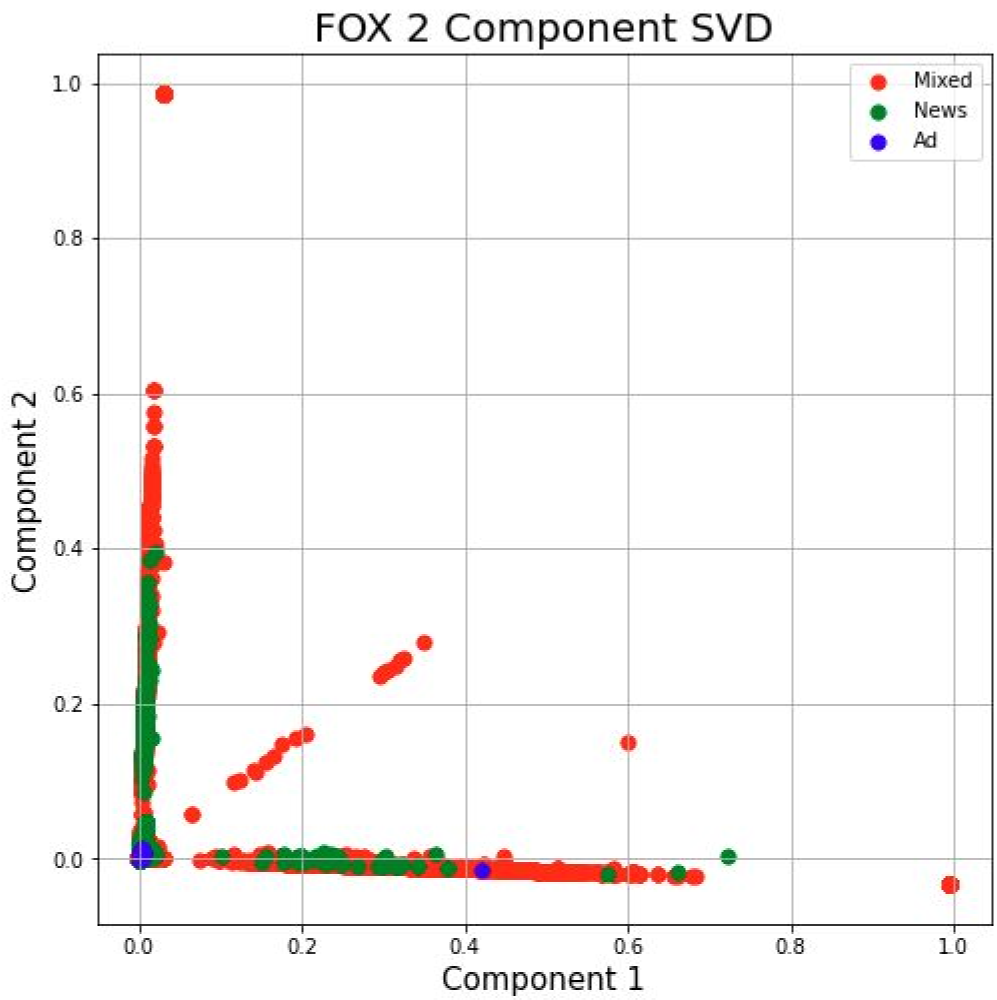
Visualize K-means

I had hoped that doing a dimension reduction on my data and visualizing the clusters might show some interesting groups. I performed Kmeans clustering on CNN sentences and coded the clusters as ad, news, or mixed. I then used TruncatedSVD to collapse the data to two dimensions and plotted it:



This doesn't provide much insight, but it is interesting to note that the news clusters do stay together, even if mixed (which are behind the news clusters) are along both axes.

For completeness, here is the same image for FOX:

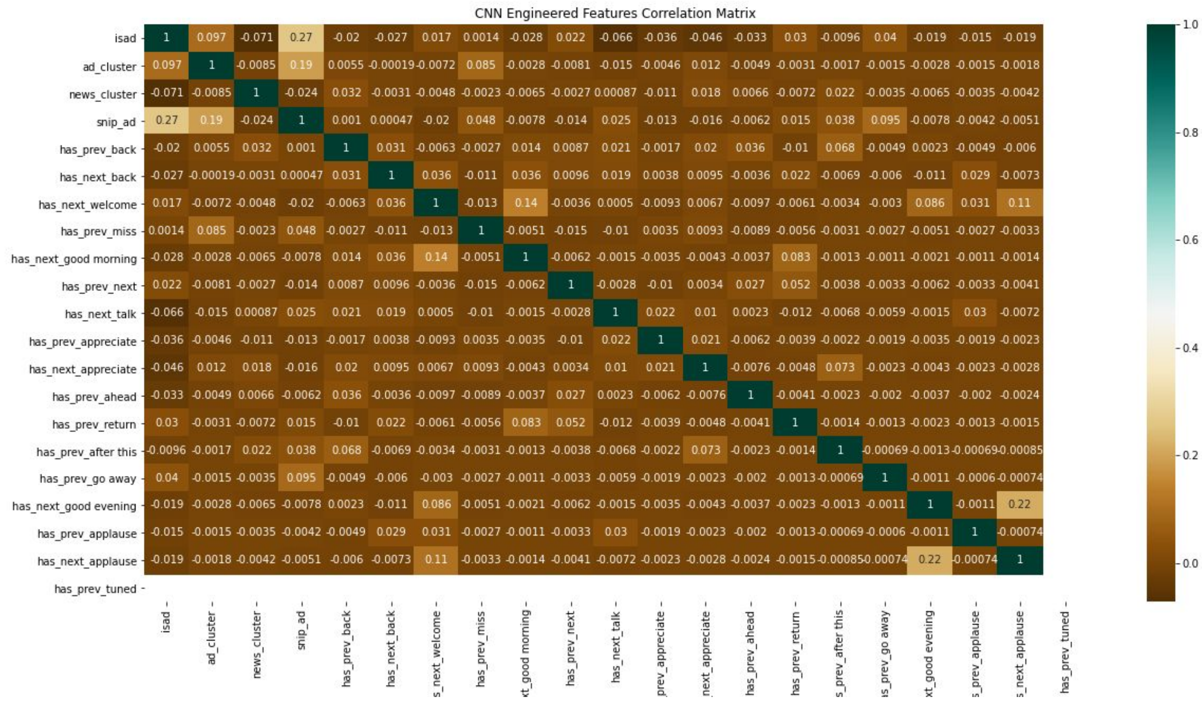


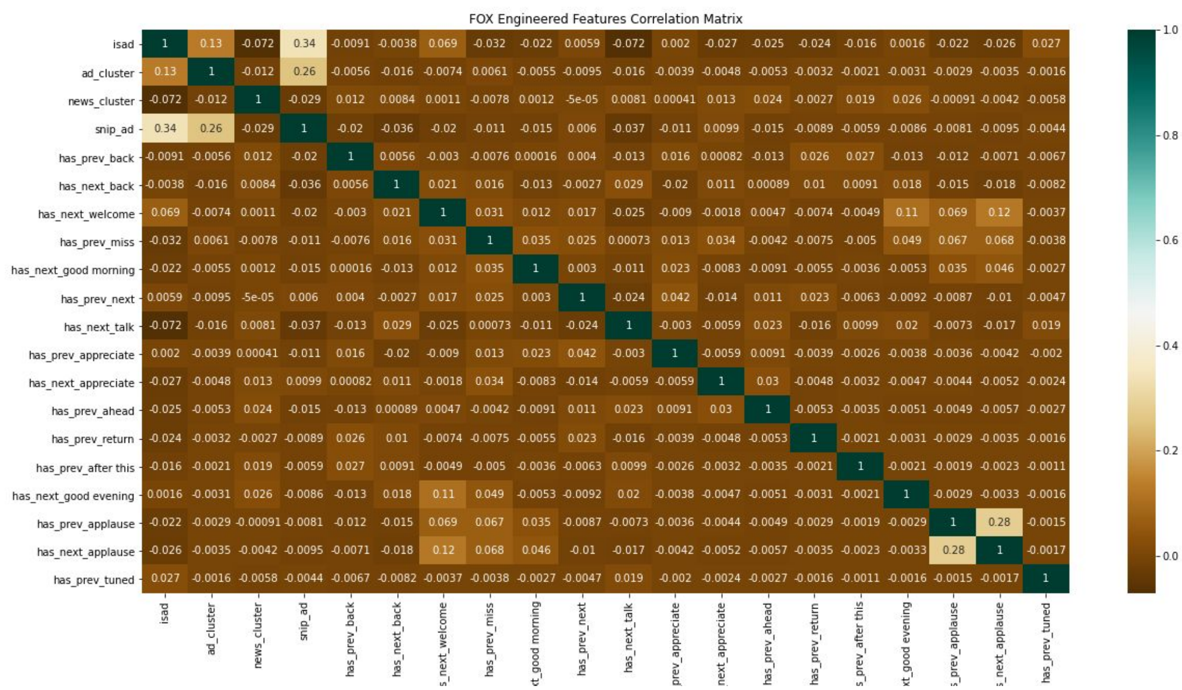
The code for this clustering and dimension reduction can be found here: https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/Clustering%20CNN%20Data.ipynb and https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/Clustering%20FOX%20Data.ipynb

Correlation Matrices

I wanted to get insight into how closely correlated my engineered features were with identified ads and also to see if there were any overlaps that were too strong between topics.

These are correlation matrices for the engineered features, showing that some at least have a good positive correlation with ads being present:





I also examined correlation matrices for all of the topics to see if any stood out as being correlated with ads, or too strongly correlated with each other. Here is an example, but they all show pretty much the same thing: weak correlation with ads and with one another:

See these notebooks for code to create correlation matrices:
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/CNN%20Correlations%20and%20Statistics.ipynb and
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/FOX%20Correlations%20and%20Statistics.ipynb

Machine Learning

Input Data

After performing the feature creation and hand coding, I used TFIDF with Lemmatization to create single words and bigrams, and appended that to the features. So the features in the machine learning inputs were:

- **Start_snip:** second at which the snippet begins
- **End_snip:** second at which the snippet ends
- **Cluster:** kmeans determined cluster
- **Ad_cluster:** whether that cluster was hand-identified as being an ad
- **News_cluster:** whether that cluster was hand-identified as being news
- **Snip_ad:** whether the snippet was diagnosed to contain ad with the first pass logistic regression
- **Topic_0 to 74:** topic score using topics identified by topic modeling

- **Has_prev_back, has_next_back, has_next_welcome, has_prev_miss, has_next_good morning, has_prev_next, has_next_talk, has_prev_appreciate, has_next_appreciate, has_prev_ahead, has_prev_return, has_prev_after this, has_prev_go away, has_next_good evening, has_prev_applause, has_next_applause, has_prev_tuned:** a set of columns which code for whether previous or following sentences have the word, for example, has_prev_back means that in the previous 2 sentences, the sentence contained the word “back”. This is helpful because news anchors often say: “We’ll be right back” before a commercial break, and “welcome back” afterwards
- A sparse matrix of the output of the TFIDF Vectorization

The labels were in a field called “isad” which I coded by hand.

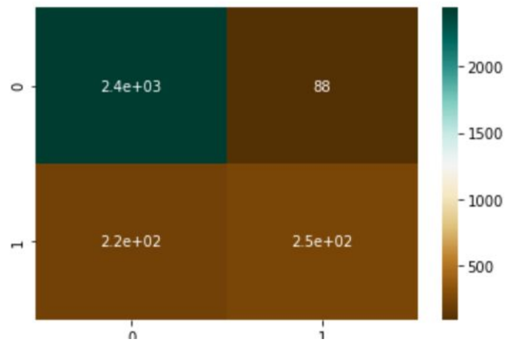
Finding Effective Models

I tested a number of classification models on the CNN and FOX data sets

Model	**Accuracy CNN**	**Accuracy FOX**
LogisticRegression	0.792	0.867
KNeighborsClassifier	0.750	0.863
SVC	0.715	0.844
LinearSVC	0.741	0.865
SGDClassifier	0.359	0.842
DecisionTreeClassifier	0.803	0.874
RandomForestClassifier	0.853	0.891
BaggingClassifier	0.822	0.899
GradientBoostingClassifier	0.786	0.880
AdaBoostClassifier	0.785	0.878

I also examined the confusion matrixes for each model and generally found them to be fairly balanced. Here is an example for the Bagging Classifier on FOX data. It generally found more false positives than negatives, but not too bad:

```
[[2445   88]
 [ 215  252]]
```



I will explore the confusion matrix for a few models more below.

See these notebooks for more information on creating these models:
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/FOX%20Supervised%20Learning.ipynb and
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/CNN%20Supervised%20Learning.ipynb

Tuning the Models

I tuned the CNN DecisionTreeClassifier, RandomForestClassifier, and BaggingClassifier using RandomizedSearchCV, since these models take so long to create. I was able to marginally improve the RandomForestClassifier, from 0.853 accuracy to 0.857.

Tuning the top FOX models (RandomForestClassifier, BaggingClassifier, GradientBoostingClassifier) with RandomizedSearchSV did not yield an improvement over the default hyperparameters. Tuning these models was also extremely time-consuming, even on an extra-large AWS instance.

See these notebooks for more details on the tuning attempted:
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/Tuning%20CNN%20Models.ipynb and
https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/Tuning%20FOX%20Models.ipynb

Feature Importance

Examining feature importance will reveal:

- Whether the created features were useful
- What words and bi-grams best predict ads

This table shows the top 20 features for predicting ads.

Feature	Importance
start_snip	0.063977726
end_snip	0.060999728
snip_ad	0.025750646
president	0.012447353
easy	0.008710522
wa	0.005392661
doctor	0.004642851
think	0.004489554
topic_3	0.004409409
topic_17	0.004370745
topic_51	0.004361802
topic_72	0.004354532
has_next_talk	0.004221578
topic_35	0.004177771
topic_49	0.004047728
topic_54	0.00404182
topic_57	0.004034425
topic_13	0.003999338
topic_25	0.00399873

Start_snip and end_snip are the number of seconds, relative to the beginning of the program, that begin and end the snippet where the sentence occurred. Since end_snip is always 60 seconds after start_snip, I probably should have dropped that column. It makes sense that these would have a high level of importance since ads occur in blocks, somewhat regularly distributed through the programs.

It also appears that doing the first pass logistic regression model to predict whether snippets had ads was also a helpful feature to add. It also appears that topic modeling, was helpful as an input to the final models.

‘President’ appears as a very predictive word, which makes sense. That word appears very frequently, and only in news segments.

Has_next_talk and has_next_back were the only engineered features regarding nearby sentences that were in the top 100 important features.

Below are the important features for predicting ads in FOX news programs.

Feature	**Importance**
start_snip	0.044676518
snip_ad	0.044558297
end_snip	0.042547359
liberty	0.006252189
oh	0.005960919
car	0.005744653
easy	0.005208301
president	0.00491346
doctor	0.004716136
topic_73	0.004566933
dad	0.004190579
hey	0.004122899
awesome	0.004028505
topic_47	0.003949441
topic_70	0.003606239
topic_30	0.003529252
topic_27	0.003528188

topic_21	0.003513364
topic_8	0.0034007

Interestingly, topics show up a little lower here, and only one of the engineered features, has_next_welcome appears in the top 100 features. Still, snippet start and end, and the snippet coding that I did with the first pass logistic regression model are important for predicting FOX ads as well.

Exploring the Models

Wild Data

I tested the best model on an additional 300 sentences of closed captioning that were not part of the testing and training set to see how the models performed on wild data. The CNN model had 95.3% accuracy on the additional 300 sentences. The FOX model had 86% accuracy on the additional 300 sentences. Those are both reasonable accuracies on a small additional test set.

CNN on FOX, FOX on CNN

I applied the FOX model to CNN data and vice versa to see how well they predicted one another. I found that the CNN model correctly predicted FOX ads 87.7% of the time, which is better than the CNN model performed on CNN sentences. The FOX model predicted CNN ads 75.7% of the time.

False Positives and False Negatives

I examined false positives and false negatives from each set. In both cases, looking at the false positives actually revealed sentences that I had miscoded in my hand-coding pass, which means that the model is better than I thought.

False negatives were often very short sentences. They also showed words, like the brand name 'chantix' which I could add to a brand-name set and use that to create a feature like 'has brand name' which might improve the model performance.

For the code to find feature importances, and the additional testing, see these notebooks: https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/FOX%20Feature%20Importance.ipynb and https://github.com/LinneaHarts/ad_finder_cc/blob/master/notebooks/CNN%20Feature%20Importance.ipynb

Conclusions and Next Steps

This project was envisioned as a proof of concept, which I believe has promise, but which also reveals some areas that need more refinement. It also showed that classification models yielded better results for FOX than CNN, likely because FOX has fewer different advertisers.

In developing this project further, I would consider:

- These models will get inaccurate in a hurry and need constant updates as new advertisers buy airtime
- With the prevalence of brand names, it would probably be helpful to do entity extraction on the sentences, code the entities as being brands, and then create a feature or features that show whether a brand name was mentioned in the ad or in a nearby sentence
- Because ads are well-predicted by the words that come before, a long short term memory deep learning neural network might be the best way to predict them
- An interactive web application that allows new hand-coding to continually improve the models and a way to upload lists of advertisers to help search for those brands might be a good way to keep the models up to date
- Removing or identifying ads is one of many NLP projects that one could do with TV news closed captions. Others include:
    - Tracing and predicting the rise and fall of news stories
    - Tracing and predicting how politicians' talking points are repeated by news anchors
    - Graphing emerging topics like twitter does with trending topics
    - Comparing vocabulary used on different networks or between different shows