FAKE NEWS CLASSIFICATION

SUPERVISED MACHINE LEARNING

AGENDA

- Data Cleaning / Preprocessing
- Exploratory Data Analysis
- Model Selection
- Oversampling
- Conclusion and Next Steps

DATA PREPROCESSING

- Load file, drop index, drop nan, drop duplicates
- Separate out the email links, web links, hashtag, mentions.
- Clean the data leakage problems
- Count the number of sentences in each article and calculate mean sentence length.
- Drop the foreign language rows as determined by langdetect
- Check for non-recognized or non-ascii characters
- Calculate the title and text similarity

WELFake.csv

	Unnamed: 0	title	text	label
0	0	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	No comment is expected from Barack Obama Membe...	1
1	1	NaN	Did they post their votes for Hillary already?	1
2	2	UNBELIEVABLE! OBAMA’S ATTORNEY GENERAL SAYS MO...	Now, most of the demonstrators gathered last ...	1
3	3	Bobby Jindal, raised Hindu, uses story of Chri...	A dozen politically active pastors came here f...	0
4	4	SATAN 2: Russia unvelis an image of its terrif...	The RS-28 Sarmat missile, dubbed Satan 2, will...	1

LABEL COUNT

```python
# show the number of rows labeled 1 and number of rows labeled 0
print(data['label'].value_counts())
```

```
label
1    37106
0    35028
Name: count, dtype: int64
```

Note on matching Real and Fake labels: In the dataset description the authors state that there are 72,134 news articles with 35,028 real and 37,106 fake news articles. The authors then go on to state the labels are labeled as follows: **0=fake and 1=real**. The two statements are contradictory based on the following label counts. Upon further inspection of the data and following the authors first statement of '35,028 real and 37,106 fake news articles', I am following the mapping for the labels as **0=real and 1=fake**.

STRIPING OUT HASHTAGS AND MENTIONS MADE THIS POSSIBLE

DATA LEAKAGE EXAMPLES

Title/Text	Examples
Title - Fake	• ['[Video]'] • ['(IMAGES/VIDEO)'] • ['(SCREENSHOTS/VIDEO)'] • ['[1 hour 8 minute video]'] • ['(NSFW VIDEO)']
Title - Real	• [' - The New York Times'] • [' - Breitbart']
Text - Real	• WASHINGTON (Reuters) - NEW DELHI (Reuters) – • (This version of the Dec 6 story corrects paragraph 3 and adds new paragraph 4 to clarify the nature of U.S. sanctions) By Patricia Zengerle WASHINGTON (Reuters) -MOSCOW (Reuters) - • ABOARD CLINTON CAMPAIGN PLANE (Reuters) -

FILTER FOR ENGLISH

Prior to filter out non-English words I counted sentences and mean length of sentences. Mean length of sentences was important to do before running langdetect because the package requires a decent sized text passage.

```
# runs for 16min
# this method will detect other languages using langdetect package
def detect_language(text):
    try:
        return detect(text)
    except:
        return 'could not detect language'
# we are running the detect_language methon on the 'text_clean' column
data.loc[:,'lang'] = data.loc[:,'text_clean'].apply(detect_language)
```

```
# the count of english slightly changes every time I run the above method but the count should be around #61608
print(data.loc[:,'lang'].value_counts())
```

```
lang
en       61604
ru         156
es         141
de          98
fr          33
ar          19
tr           7
pt           7
it           5
hr           4
nl           3
no           3
pl           2
el           2
sq           1
zh-cn        1
vi           1
sw           1
Name: count, dtype: int64
```

TITLE & TEXT SIMILARITY

```
# make a trained word2vec model from the title and text tokens
# vector_size is the dimensionality of the word - the higher dimension can capture more complex relationships
# window determines how many words back and forward to look around the word
word2vec_model = Word2Vec(tokens_collection, vector_size=300, window=5, min_count=2, workers=4)
```

```
def title_text_similarity(title, text, model=word2vec_model):
  title_tokens = title.split()
  text_tokens = text.split()

  #the title_vec and text_vec need to be at same size as vector_size in the word2vec_model
  title_vec = np.zeros(300)
  text_vec = np.zeros(300)

  # loop through title tokens
  for token in title_tokens:
    # if token in word2vec model add to title_vec
    if token in model.wv:
      title_vec = np.add(title_vec, model.wv[token])

  # loop through text tokens
  for token in text_tokens:
    # if token in word2vec model add to title_vec
    if token in model.wv:
      text_vec = np.add(text_vec, model.wv[token])


  # if either title_vec or text_vec is a zero vector return 0
  if np.linalg.norm(title_vec) == 0 or np.linalg.norm(text_vec) == 0:
      return 0  # or any other default value you prefer
  else:
      # similarity calculation = (dot product of title_vec and text_vec) / (magnitude of title_vec *
      # cosine similarity is calculated by dividing the dot product of two vectors by the product of
      return round(np.dot(title_vec, text_vec) / (np.linalg.norm(title_vec) * np.linalg.norm(text_ve
```

I did expect the real articles to have a higher title to text similarity score

```
# check how title_text_similarity compare based on label
print(data.loc[data['label']==0, 'title_text_similarity'].mean())
print(data.loc[data['label']==1, 'title_text_similarity'].mean())
```

```
0.6771524150681766
0.6432999629382551
```

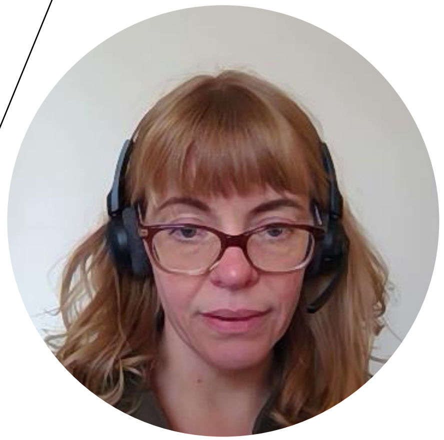
Filtered non-words and non-ascii strings out.

RESULTS FOR CHECKING NON-WORD PERCENTAGE

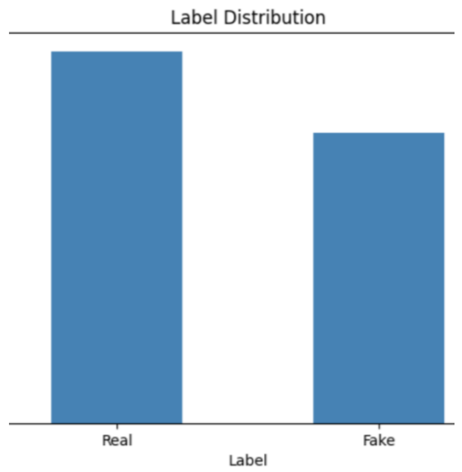
I did expect the real articles to have a lower non-word percentage

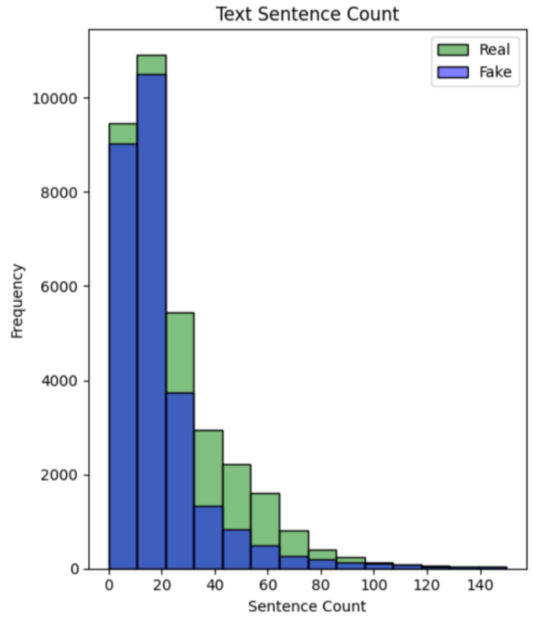
```
# compare non_word_percent by label
print(data.loc[data['label'] == 0, 'non_word_percent'].mean())
print(data.loc[data['label'] == 1, 'non_word_percent'].mean())
```

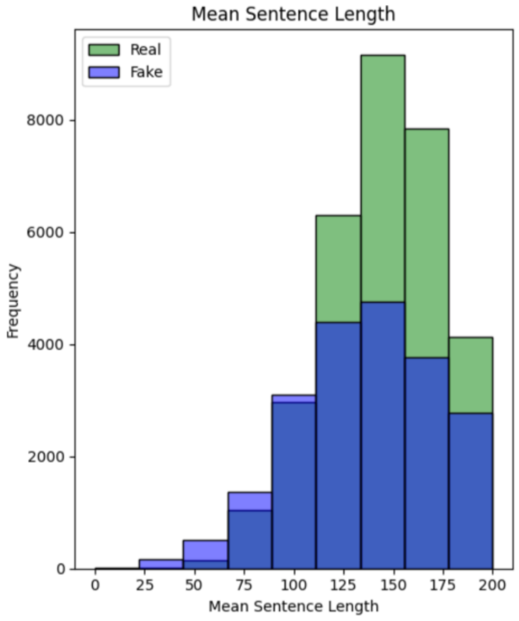
```
0.13381513238181678
0.19963963905746687
```

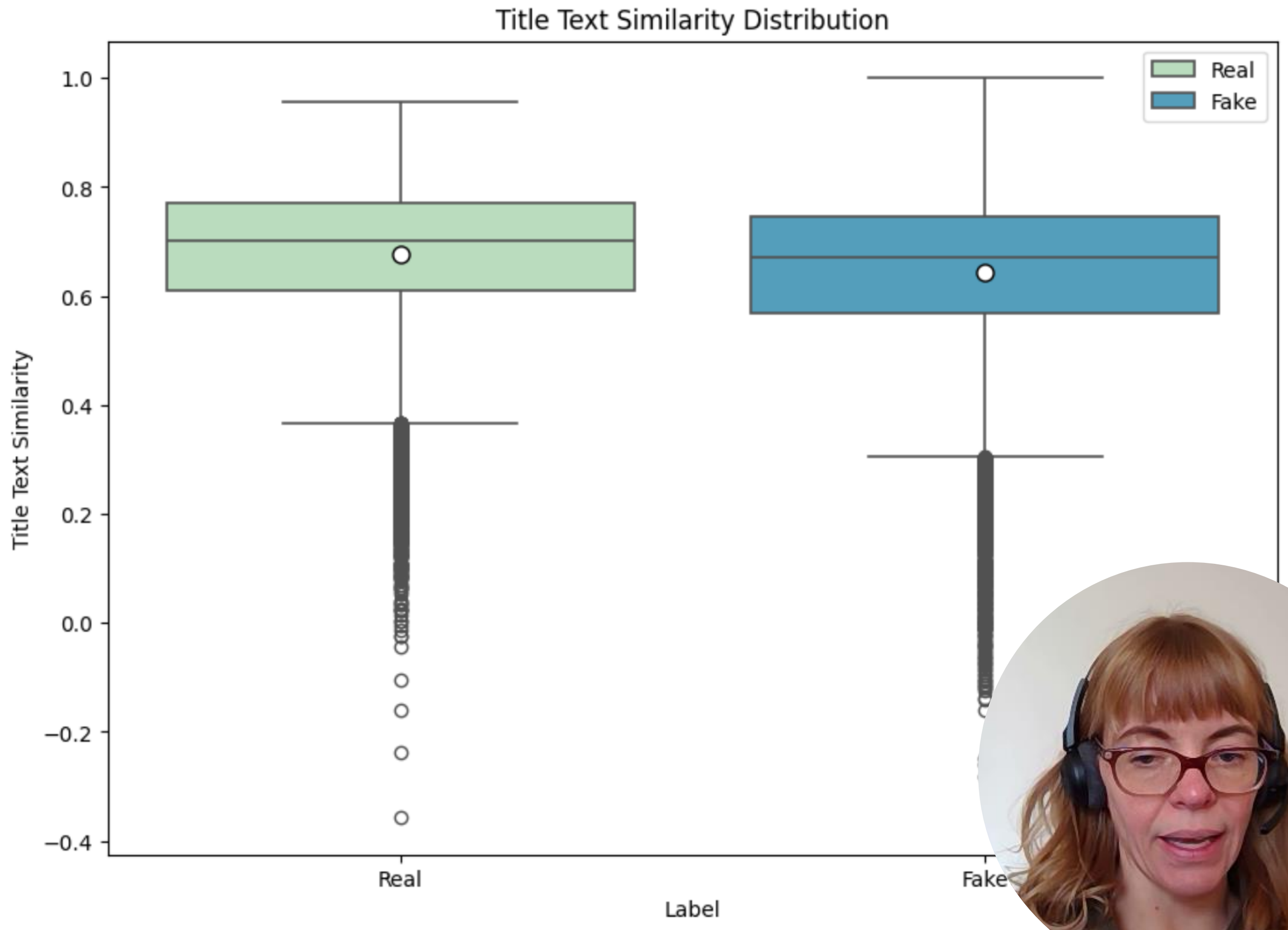


EDA – LOOKING AT LABEL, SENTENCE, AND MEAN LENGTH DISTRIBUTION

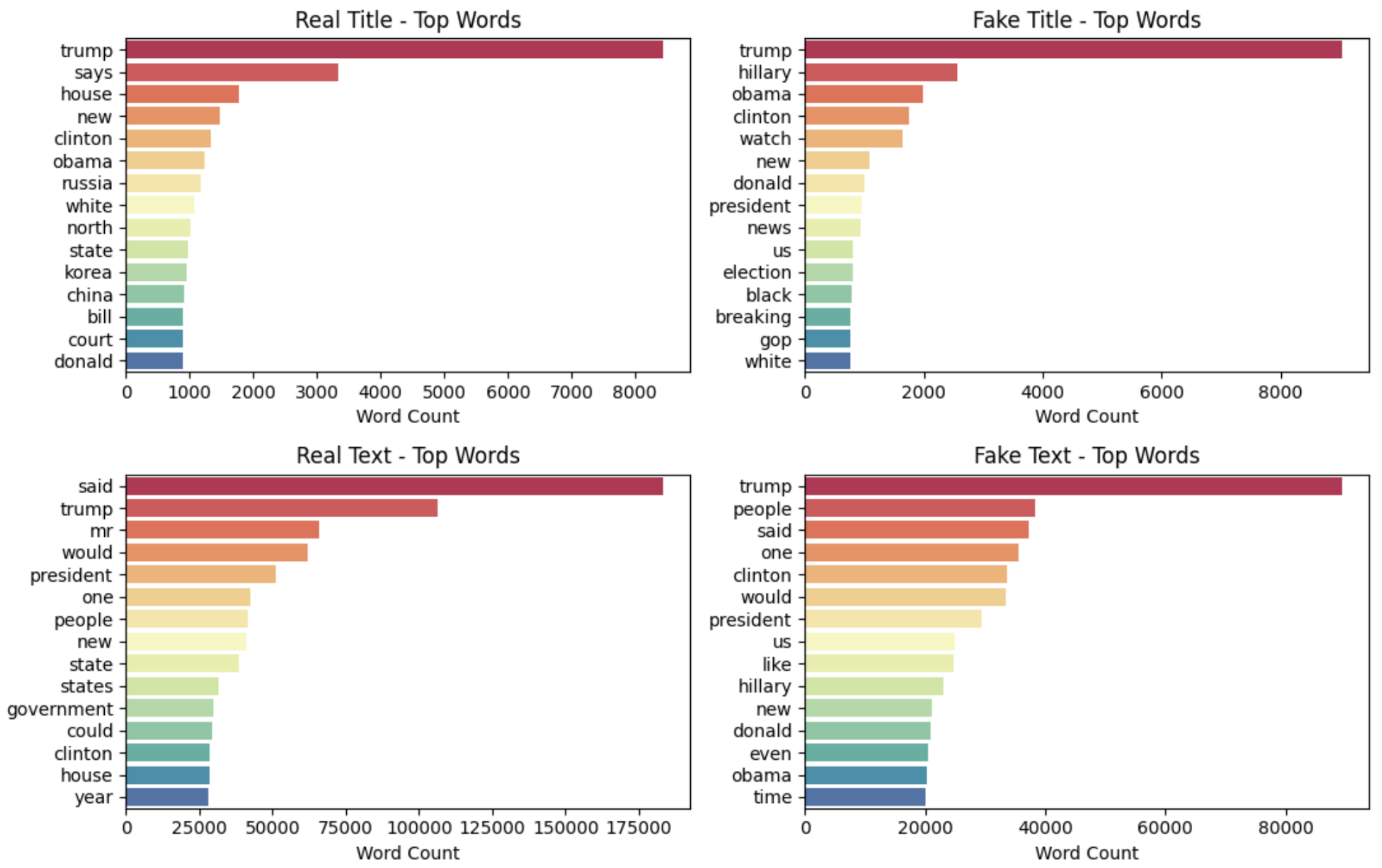






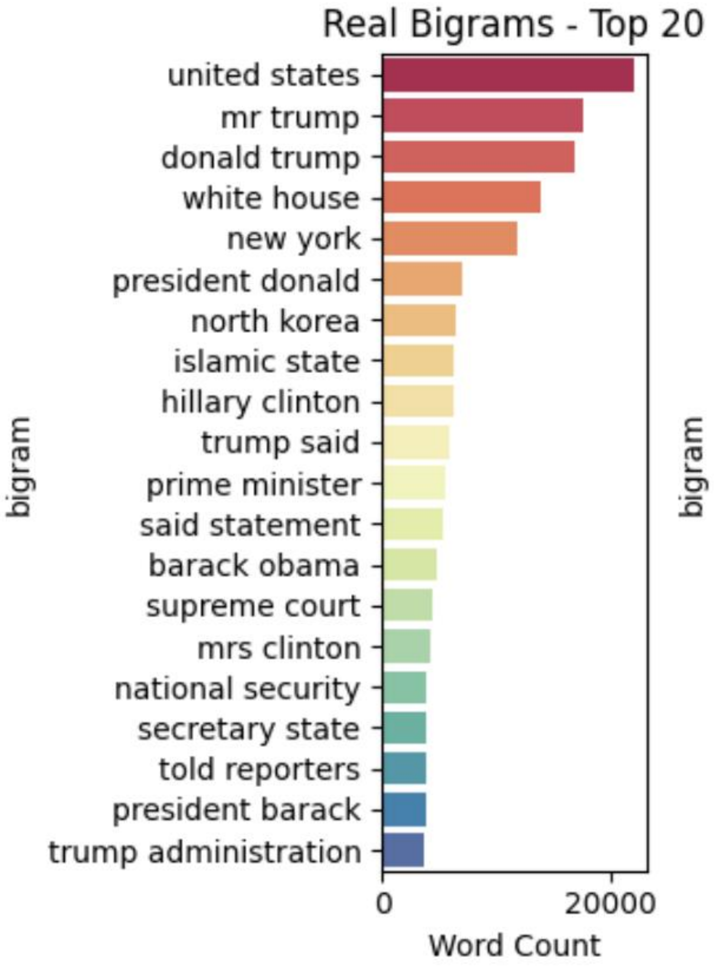


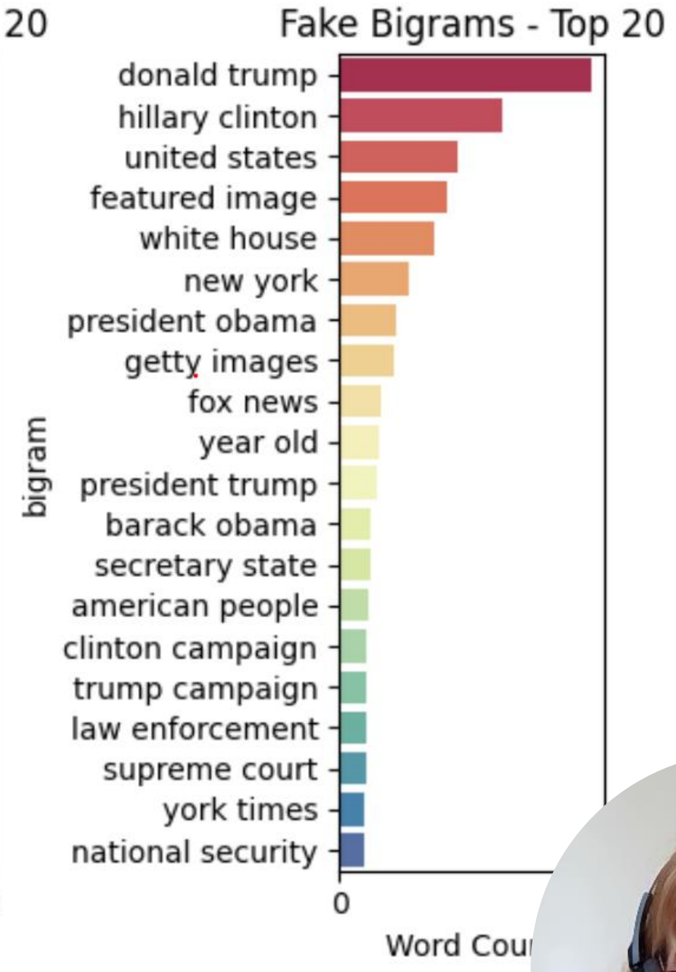
TOP WORDS IN TITLES AND TEXTS



TOP BIGRAMS

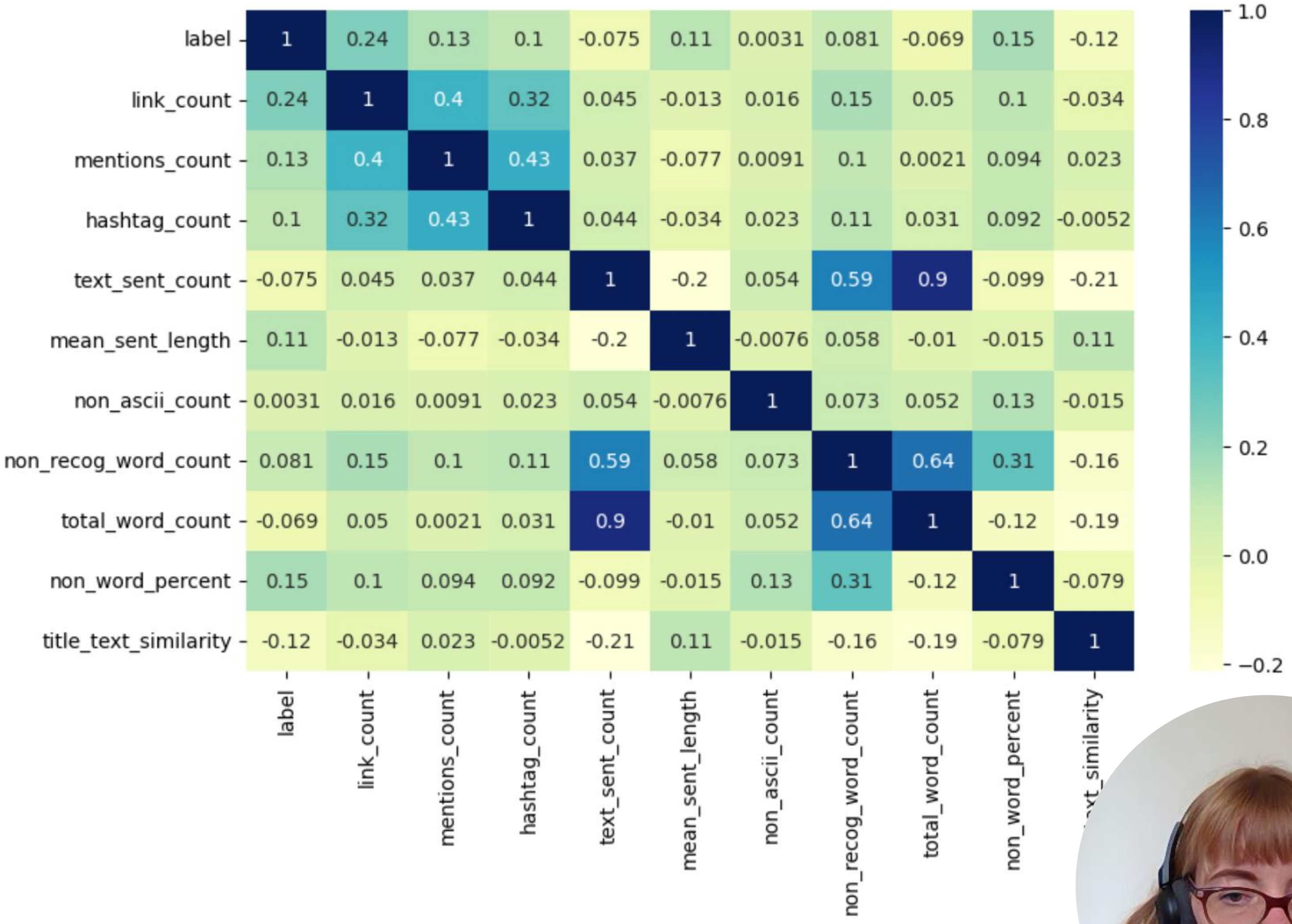
Bigrams – pair of consecutive words in written text

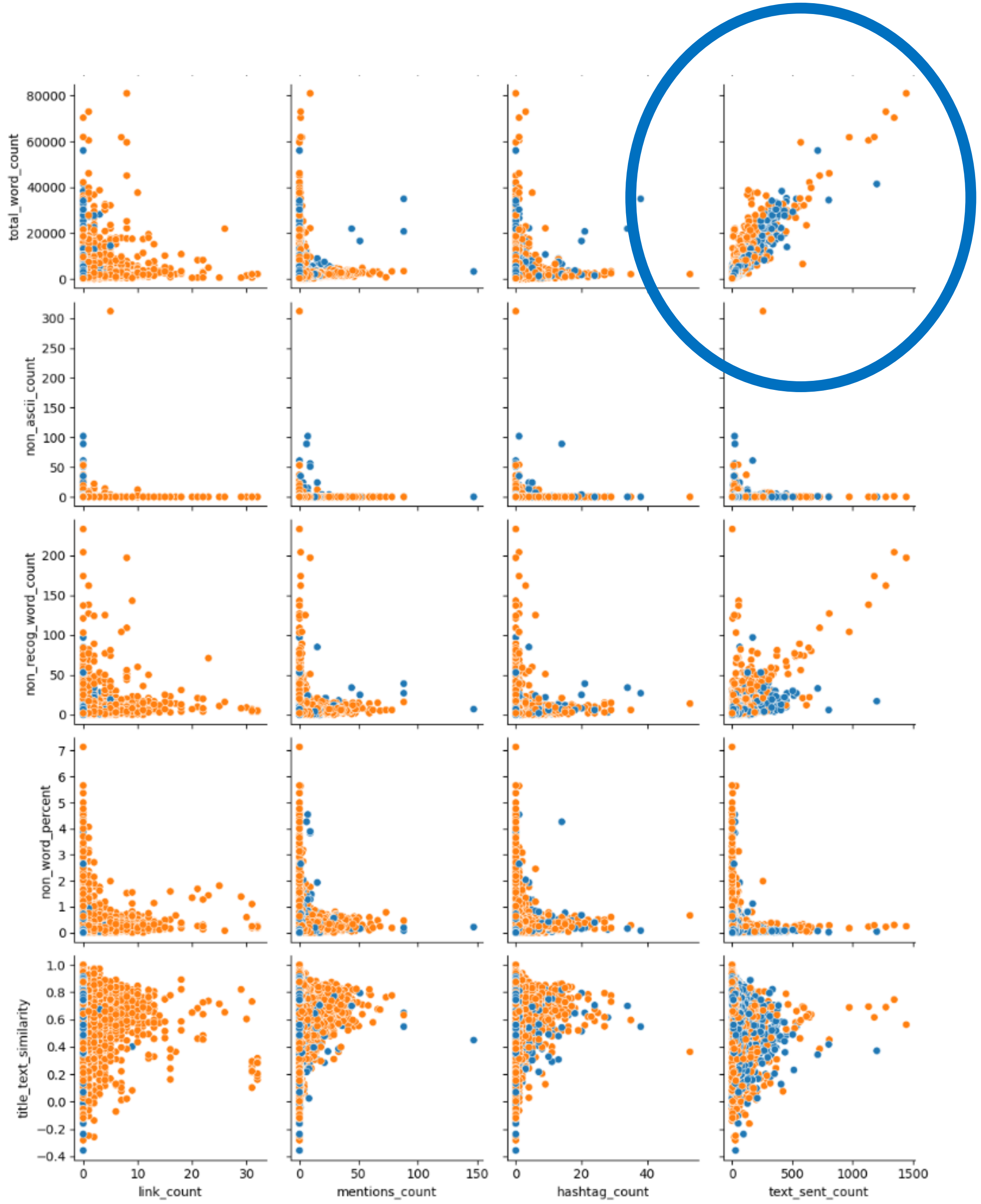




CORRELATION MATRIX

I am only concerned with the ‘text_sent_count’ and ‘total_word_count’ colinear relationship





PAIR PLOT

I am only concerned with the ‘text_sent_count’ and ‘total_word_count’ colinear relationship

FEATURE SELECTION

List of non-text features

1	link_count	61598	non-null	int64
2	mentions_count	61598	non-null	int64
3	hashtag_count	61598	non-null	int64
4	mean_sent_length	61598	non-null	float64
5	non_ascii_count	61598	non-null	int64
6	non_recog_word_count	61598	non-null	int64
7	total_word_count	61598	non-null	int64
8	non_word_percent	61598	non-null	float64
9	title_text_similarity	61598	non-null	float64

Code

```
# create a decision tree classifier
estimator = DecisionTreeClassifier()

# use RFE with cross-validation to find best number of features
selector = RFECV(estimator, cv=5)
selector = selector.fit(X_train.drop('text_clean_ascii', axis=1), y_train)

# display best number of features
print("Optimal number of features: %d" % selector.n_features_)

# display selected features
print("Selected features: %s" % selector.support_)
```

```
Optimal number of features: 9
Selected features: [ True  True  True  True  True  True  True  True  True]
```

WORD2VEC VECTORIZATION

We chose to download the word2vec google news model and then write a method to use the prebuilt model to vectorize our dataframe.

Google News model contains 300-dimensional vectors for 3 million words and phrases.

```
[21] # method to handle errors in downloading word2vec google new model
     def download_w2v_model(model_name):

       attempts = 0
       max_attempts = 3
       # loop to try to load google news model
       while attempts < max_attempts:
         try:
           model = api.load(model_name)
           return model

         # if there is an exception add to attempts count and try again
         except Exception as e:
           print(f"Error downloading {model_name}: {e}")
           attempts += 1
           print(f"Download attempt {attempts} failed. Retrying...")

       print('Failed to download')
       return None

     # load google news word2vec model
     model_google = download_w2v_model("word2vec-google-news-300")
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

```
[22] # method for w2v vertorization with pretrained model
     def w2v_vectorize(text, model):
       words = text.split()
       # word is in vocabulary retrieve corresponding word embedding
       words_vecs = [model[word] for word in words if word in model]
       # placeholder for text with no recognized words
       if len(words_vecs) == 0:
         return np.zeros(100)
       # convert to np array
       words_vecs = np.array(words_vecs)
       # returns vector that is the averages of words_vecs
       return words_vecs.mean(axis=0)
```

COLUMN TRANSFORMER

```
# Convert all column names to strings
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)

# set up the columntransformer which helps scale the non-text data
# we will passthrough on the word embedding data
preprocessor = ColumnTransformer(
    transformers = [
        ('link_count', MinMaxScaler(), ['link_count']),
        ('mentions_count', MinMaxScaler(), ['mentions_count']),
        ('hashtag_count', MinMaxScaler(), ['hashtag_count']),
        ('mean_length', MinMaxScaler(), ['mean_sent_length']),
        ('non_ascii', MinMaxScaler(), ['non_ascii_count']),
        ('non_recog', MinMaxScaler(), ['non_recog_word_count']),
        ('total_words', MinMaxScaler(), ['total_word_count']),
        ('non_word_per', MinMaxScaler(), ['non_word_percent']),
        ('similarity', MinMaxScaler(), ['title_text_similarity']),
    ],
    remainder='passthrough'  # This keeps 'word2vec' unchanged
)
# fit and transform the train data and transform the test data
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# run the models with the data
run_models(X_train_transformed, y_train, X_test_transformed, y_test)
```

CHOICE OF MODELS – CLASSIFICATION

Logistic Regression

- Efficient with large dataset
- Not sensitive to feature scaling

SGD Classifier

- Fast to train and good with large datasets
- Supports online learning

Random Forest

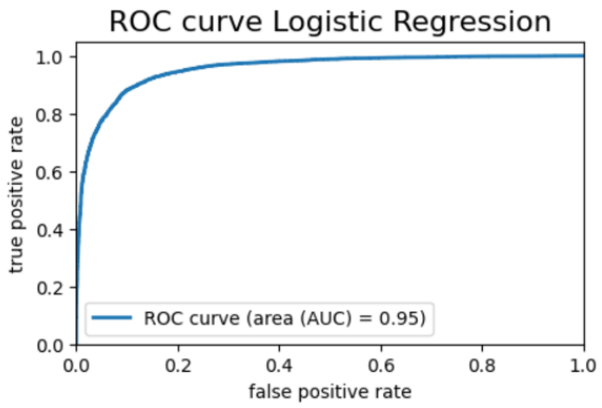
- Handles non-linear relationships
- Performs better on dense data which is why it is important that we use word2vec vectorization method

SVM

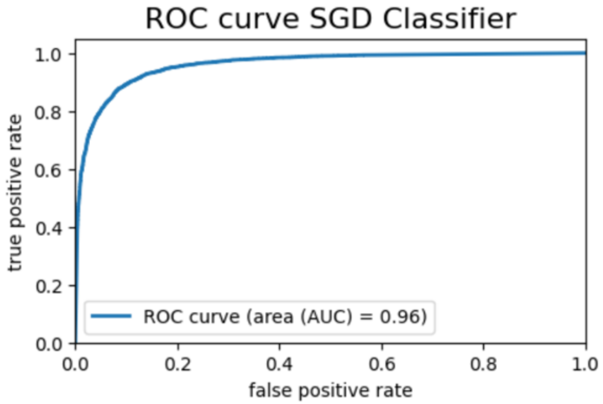
- Good with high-dimensional text data
- Can adapt to different types of data distributions

BEFORE OVERSAMPLING

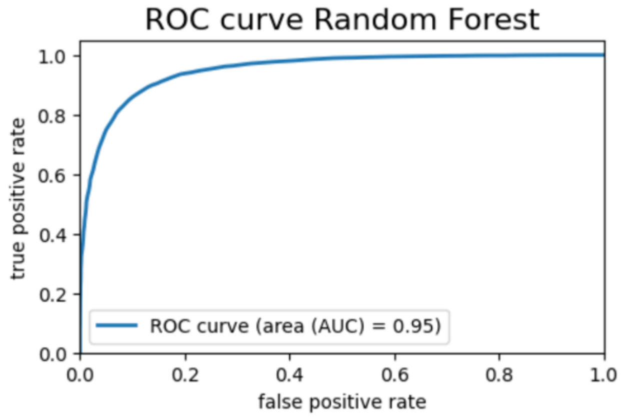
```
Logistic Regression Accuracy: 0.8906 Time: 0.33 seconds
              precision    recall  f1-score   support

           0       0.90      0.91      0.90      8688
           1       0.88      0.86      0.87      6712

    accuracy                           0.89     15400
   macro avg       0.89      0.89      0.89     15400
weighted avg       0.89      0.89      0.89     15400
```



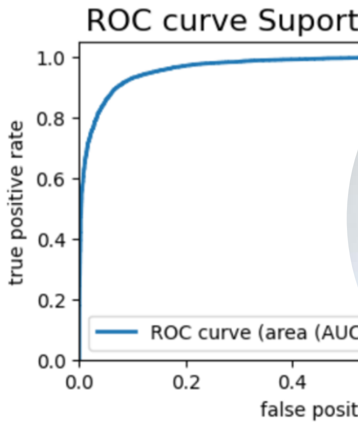

```
SGD Classifier Accuracy: 0.8984 Time: 46.85 seconds
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      8688
           1       0.89      0.88      0.88      6712

    accuracy                           0.90     15400
   macro avg       0.90      0.90      0.90     15400
weighted avg       0.90      0.90      0.90     15400
```




```
Random Forest Accuracy: 0.8813 Time: 120.61 seconds
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      8688
           1       0.87      0.85      0.86      6712

    accuracy                           0.88     15400
   macro avg       0.88      0.88      0.88     15400
weighted avg       0.88      0.88      0.88     15400
```



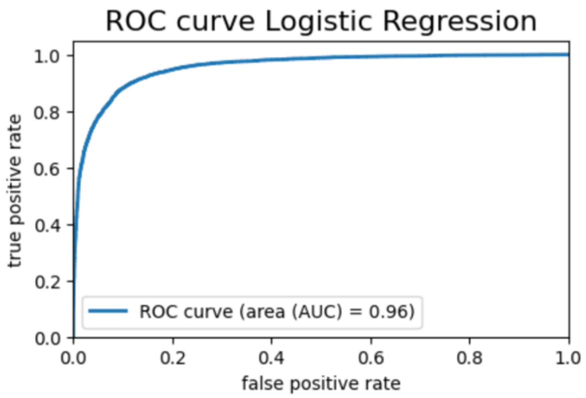

```
Suport Vector Machine Accuracy: 0.9152 Time: 1222.24 seconds
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      8688
           1       0.91      0.90      0.90      6712

    accuracy                           0.92     15400
   macro avg       0.91      0.91      0.91     15400
weighted avg       0.92      0.92      0.92     15400
```



AFTER OVERSAMPLING

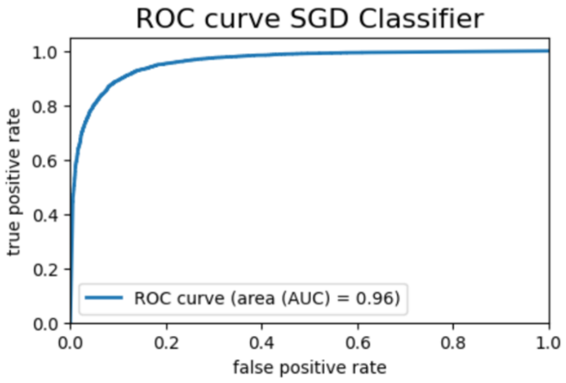
Logistic Regression Accuracy: 0.8912 Time: 0.29 seconds

	precision	recall	f1-score	support
0	0.91	0.90	0.90	8688
1	0.87	0.88	0.88	6712
accuracy			0.89	15400
macro avg	0.89	0.89	0.89	15400
weighted avg	0.89	0.89	0.89	15400



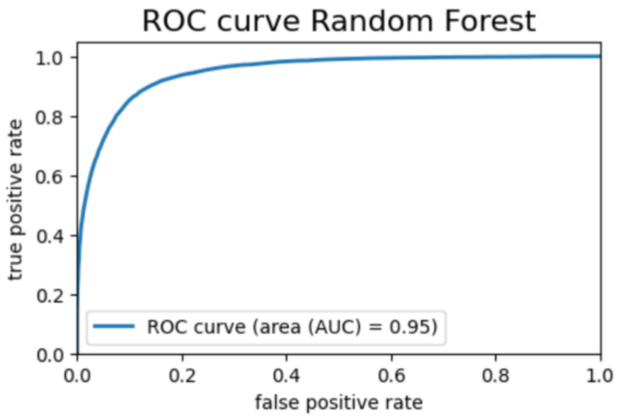
SGD Classifier Accuracy: 0.8964 Time: 52.17 seconds

	precision	recall	f1-score	support
0	0.92	0.90	0.91	8688
1	0.87	0.89	0.88	6712
accuracy			0.90	15400
macro avg	0.89	0.90	0.89	15400
weighted avg	0.90	0.90	0.90	15400



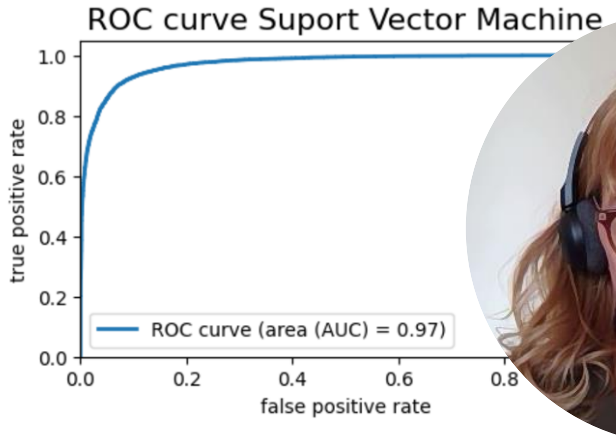
Random Forest Accuracy: 0.8801 Time: 132.96 seconds

	precision	recall	f1-score	support
0	0.89	0.89	0.89	8688
1	0.86	0.86	0.86	6712
accuracy			0.88	15400
macro avg	0.88	0.88	0.88	15400
weighted avg	0.88	0.88	0.88	15400



Suport Vector Machine Accuracy: 0.9156 Time: 1539.90 seconds

	precision	recall	f1-score	support
0	0.93	0.92	0.92	8688
1	0.90	0.91	0.90	6712
accuracy			0.92	15400
macro avg	0.91	0.92	0.91	15400
weighted avg	0.92	0.92	0.92	15400



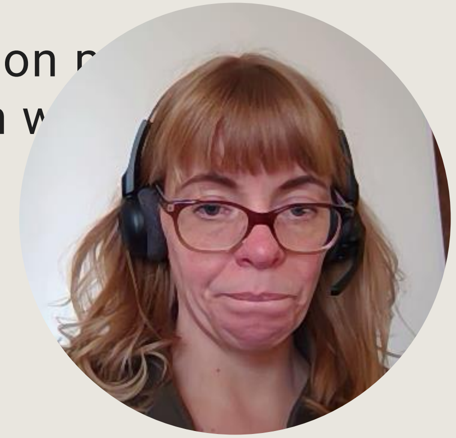
FINAL TIPS & TAKEAWAYS

Conclusion – Chose generalizability over accuracy scores.

I learned a lot from this project. Initially, I used a TF-IDF vectorizer, which gave me higher accuracy scores. However, with around 200,000 features, I believed the model was overfitting to the data. This insight prompted me to investigate the word2vec word embeddings. My scores went down but my models were more reliable, and I think more predictive of current new articles. I chose prioritizing generalizability over optimizing accuracy.

Next Steps:

- Work on parameter tuning – my computer kept crashing when using gridsearch
- Use my knowledge gained on word2vec and play around more with context and meaning.
- I want to create a book recommendation m
and I think the knowledge gained from w
on this project will help me.



THANK YOU

Desiree Disco

https://github.com/desireedisco/MSDS-Machine-Learning-Supervised